A DISTRIBUTED MONTE CARLO METHOD FOR INITIALIZING STATE VECTOR DISTRIBUTIONS IN HETEROGENEOUS SMART SENSOR NETWORKS

A Dissertation
Presented to
The Academic Faculty

By

Milind Borkar

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
April 2008

A DISTRIBUTED MONTE CARLO METHOD FOR INITIALIZING STATE VECTOR DISTRIBUTIONS IN HETEROGENEOUS SMART SENSOR NETWORKS

Approved by:

Dr. James H. McClellan, Committee Chair
Professor, School of ECE
Georgia Institute of Technology

Dr. Bonnie Heck Ferri
Professor, School of ECE
Georgia Institute of Technology

Dr. David V. Anderson
Associate Professor, School of ECE
Georgia Institute of Technology

Dr. Brani Vidakovic
Professor, School of BME
Georgia Institute of Technology

Dr. Monson H. Hayes
Professor, School of ECE
Georgia Institute of Technology

Date Approved: 10 December, 2007

Alone we can do so little, together we can do so much.

- Helen Keller

This dissertation is dedicated to my family,

Anil, Gauri and Amol Borkar:

Without you, I would not be where I am today.

ACKNOWLEDGMENTS

As I approach the end of my long journey through Georgia Tech, I can not forget all the people who have been influential in bringing me where I am today. There is no way I could give the deserved thanks to each and every person that has affected my life, as that would end up being a book by itself. However, I will acknowledge those who have made a significant impact on me as a researcher, and more importantly, as a man.

The first set of people that I would like to thank is my family. My father, Anil Borkar, has been a driving force in my life since the day I was born. He has always encouraged me to excel and has given me every opportunity I could ask for, the never ending sequence of which has brought me to this important stage in my life, achieving the Ph.D. degree. My mother, Gauri Borkar, made my life her priority and put her own career behind. She taught me to fight for what I believed in, and to never give up. My brother, Amol Borkar, has always supported every decision I made and stood behind me during my times of need.

My decision to pursue the Ph.D. degree was strongly influenced by my advisor, Dr. James McClellan. I could not ask for a better advisor. Rarely do I meet people like him who know something about everything and who are really passionate about their profession. People like him make me want to strive for excellence. He did an amazing job of pointing me in the right direction, while at the same time giving me enough freedom to find my own path.

Another member of the faculty to whom I feel very close is Dr. Monson Hayes. He has always been very easy to approach and talk to about anything on my mind. He is a very creative thinker, and being in his company has helped boost my own creativity. I think of him as my second advisor, but more importantly, as a good friend.

I would like to thank the remaining members of my committee, namely Dr. David Anderson, Dr. Bonnie Heck Ferri, and Dr. Brani Vidakovic, for taking time out of their busy schedules to be a part of this important landmark in my life.

I have met some of the most intelligent people from all over the world at CSIP. One outstanding person is Volkan Cevher. He is a great teacher and was responsible for bringing me up to speed with Monte Carlo methods, the core of this dissertation. I have always considered him as my technical mentor and I thank him for taking the time to show me the ropes. I also want to thank the others members of our research group who have consistently provided excellent feedback and have helped improve my research: Mubashir Alam, Ali Cafer Gurbuz, Greg Krudysz, Yeongseon Lee, Qiang Li, Sam Li, Rajbabu Velmurugan and Faisal Shah. Others at CSIP that deserve thanks: Martin Tobias and Will Leven for the never ending discussions on particle filters and tracking, and Nicolas Gastaud for helping me figure out LaTeX.

I will always be grateful to the ECE administrative staff, in particular Christy Ellis, Lisa Gardner, Catherine Gholson, Kay Gilstrap, Marilou Mycko and Tammy Scott, who made sure that all my administrative issues were taken care of. Thank you all for helping things move smoothly.

Like a lot of engineers, most of my friends are males. However, I have three very close female friends who have shaped various aspects of my personality. I want to thank them in the order in which they entered my life: Anju Treohan, for teaching me to be a go-getter, Erika Diniz, for showing me that there never needs to be an end to learning, and Kimberly Stalker, for always supporting and encouraging me in every situation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The objective of this research is to demonstrate how an underlying system's state vector distribution can be determined in a distributed heterogeneous sensor network with reduced subspace observability at the individual nodes. We show how the network, as a whole, is capable of observing the target state vector even if the individual nodes are not capable of observing it locally. The initialization algorithm presented in this work can generate the initial state vector distribution for networks with a variety of sensor types as long as the measurements at the individual nodes are known functions of the target state vector. Initialization is accomplished through a novel distributed implementation of the particle filter that involves serial particle proposal and weighting strategies, which can be accomplished without sharing raw data between individual nodes in the network. The algorithm is capable of handling missed detections and clutter as well as compensating for delays introduced by processing, communication and finite signal propagation velocities. If multiple events of interest occur, their individual states can be initialized simultaneously without requiring explicit data association across nodes. The resulting distributions can be used to initialize a variety of distributed joint tracking algorithms. In such applications, the initialization algorithm can initialize additional target tracks as targets come and go during the operation of the system with multiple targets under track.

CHAPTER 1

INTRODUCTION

Sensors and sensor network research provide substantial benefits for a broad range of applications related to national security, environmental monitoring, and health care. For example, with the use of acoustic, video, RADAR, LADAR, and biosensors, we can monitor traffic, navigate robots, collect information on animal habits, understand the biological triggers and effects of cardiovascular diseases, and decode human brain signals. The current trend in sensor network research emphasizes the deployment of heterogeneous sensors operating under assorted sensor modalities to overcome the shortcomings of individual sensor modalities. In addition, sensor mobility is becoming an important issue, for example in mine detection or space exploration, to reduce redundancy in sensor networks by improved allocation of resources.

In the literature, there is extensive research on the collection of sensor data, their analysis and interpretation, and the overall design of sensor networks for optimal detection, tracking, and classification [1]. For example, acoustic sensors can be used to cue video cameras by deciding if their ambient recordings exhibit any periodicity, indicating the presence of vehicles [2, 3]. Then a vision-tracking algorithm within the camera can determine the position of the vehicle in its image plane, update the vehicle's appearance, and infer its structure using its planar motion, thereby achieving simultaneous classification [4, 5]. With the collective knowledge of many sensors with multiple modalities, the sensor network can then report the targets' states, their identities, and other parameters of interest to the sensor network operator in a robust and accurate manner. There are numerous methods available for solving the tracking problem, depending on how the noise enters the system and whether the evolution of the system is linear [6] or nonlinear [7].

Within this detection and tracking circuit, the initial estimate of the state vector is typically assumed to be known. Note that tracking provides a means to reduce computation

by focusing the current search space for the phenomena of interest, i.e., target states, near previous values in a recursive structure. However, to get started, tracking requires an initial state distribution or initialization. Moreover, for sensor network scalability, e.g., in numbers, initialization must be achieved under communication constraints. This is because the sensors in large networks tend to operate in a distributed manner as their data can only be transmitted via ad-hoc wireless channels in short distances using point-to-point communications [8, 9].

The objective of this research is to demonstrate how an underlying system's state vector distribution, which we refer to as the target state vector distribution, can be jointly determined in distributed heterogeneous sensor networks with reduced subspace observability at the individual nodes. We show how the network, as a whole, can be made capable of observing the target state vector even if the individual nodes are not capable of observing it locally. The presented algorithm can generate the initial state vector distribution for networks with a variety of sensor types as long as the measurements at the individual nodes are known functions of the target state vector. These joint inference problems tend to be highly nonlinear and non-Gaussian in nature, making analytical solutions complicated, if not impossible, to derive. The initialization algorithm presented herein uses Monte Carlo methods to produce discrete approximations to the distribution of interest. The discrete approximations are made using particles, representing hypothesized target states, and associated weights, representing the degree of belief in the particles. The initialization algorithm uses importance sampling [10] and is based on message passing between neighboring nodes under the constraint of fixed communication bandwidth. The proposed algorithm is capable of handling missed detections and clutter as well as issues resulting from delays introduced by processing, communication, and finite signal propagation velocities. If multiple events of interest, which we refer to as targets, occur, their individual states can be initialized simultaneously without requiring explicit data association across nodes. The

examples provided in this document focus on surveillance scenarios in which multiple maneuverable vehicles are moving within a sensor network. However, the methodology is very general and can be easily applied to arbitrary systems that rely on distributed processing with heterogeneous sensors. The resulting distributions can be used to initialize a variety of distributed joint tracking algorithms (DJT). In such applications, with minimal interaction with the distributed tracker, the initialization algorithm can initialize additional target tracks as targets come and go during the operation of the system with multiple targets under track.

This chapter begins with a brief overview of various sensor network configurations and a discussion of the advantages and disadvantages of each, followed by an overview of the components in our smart sensor nodes. Next, the stochastic filtering problem, which deals with estimating the hidden state parameters of a system, is introduced. The initialization problem is a special case of the stochastic filtering problem in which prior state information is not available. Monte Carlo methods are presented as attractive tools to solve the stochastic filtering problem, especially in distributed sensor network implementations. We then discuss the basic principles of importance sampling and lay the framework for the distributed initialization algorithm presented in Chapters 2 and 3. Next, we briefly discuss kernel density estimation, a nonparametric density estimation method which allow us to approximate a continuous probability density function (pdf) using a discrete probability mass function (pmf). Finally, we introduce a novel method for weighting and resampling that allows us to effectively fuse data from multiple sensor nodes.

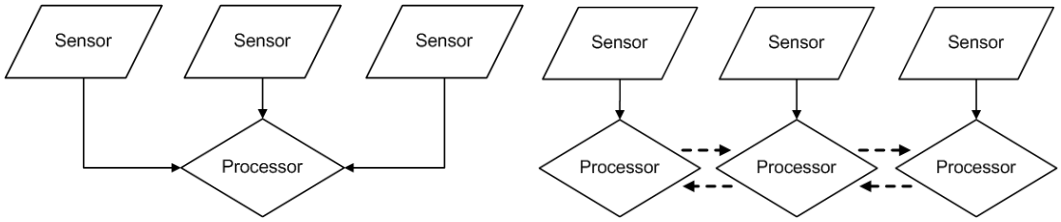
1.1 Sensor Networks

Sensor networks consist of a collection of sensor nodes distributed over the system or environment to be monitored. Based on the types of sensors in the network, these can be classified into two categories: (i) homogeneous sensor networks in which all sensor nodes

are identical to one another and (ii) heterogeneous sensor networks in which arbitrary sensor nodes observe different modalities of the system. Since all the sensors share the same modality in homogeneous sensor networks, fusing data from different sensors is relatively straightforward. This typically involves processing data that is recorded at distinct temporal or spatial locations to generate improved estimates and higher dimensional state space observability compared to those provided by a single sensor. This area has been explored in great depth in the array processing literature. Processing data in heterogeneous sensor networks is significantly more complicated than in the homogeneous case. Measurements from different modalities have to be fused together effectively even if each sensor node observes a different subspace of the target state space. State observability is another issue since each individual sensor node may only be able to observe a reduced subspace of the target state. In surveillance applications, another issue arises when the different modalities have varying signal propagation velocities. Some subset of sensor nodes, e.g., acoustic sensors, may contribute delayed information about the various targets' states, hence leading to biased estimates. This effect is magnified when the targets are moving with high velocities, are at a large range from the sensing node, or are maneuvering. Time delays may also be introduced in the system by processing and communication latencies, and these may be different at different nodes.

However, heterogeneous sensor networks have advantages over their homogeneous counterparts [11]. For example, if the sensor network operates in a single modality, then it is possible for that modality to miss the detection of an event of interest. As the number of modalities in the network increases, it progressively becomes less likely that the event of interest remains undetected by all sensor modalities. Intuitively, it is clear that the probability of detection by the sensor network will be greater than, or at least equal to, that provided by the sensor modality with the highest probability of detection present in the network. This is because different modalities may operate independent of one another and provide complementary information.

Sensor networks can also be classified based on the processing scheme employed. In *centralized sensor networks*, data from all sensor nodes is transmitted to a central processing unit that fuses the incoming information. In *distributed sensor networks*, multiple processing units exist in the network, each one processing data received from some subset of the sensors in the network to produce local estimates, and sharing statistics between each other to produce global estimates. Block diagrams representing centralized and distributed processing are given in Figure 1. In the extreme case of fully distributed processing, each sensor node may have its own dedicated processor, leading to the definition of *smart sensors*. A smart sensor is a node that not only has the ability to sense the environment but also has the ability to process incoming data and communicate with neighboring nodes.



(a) Centralized (b) Distributed

Figure 1. Centralized vs. distributed processing. The solid lines represent raw data whereas the dashed lines represent sufficient statistics.

Even though centralized approaches have the potential to arrive at the globally optimum solution, distributed sensor networks have significant advantages over centralized networks. In a centralized network, if the processing unit is incapacitated, the entire system ceases operation. This is not a problem in distributed sensor networks because all nodes are involved in processing data. Communication is another issue in centralized networks. Transmitting raw data from each sensor node to the central processor requires tremendous bandwidth as the number of sensors increases. Communication typically requires more power than processing [12] and this power requirement increases as the range for communications increases ($\propto r_t^4$). Thus, centralized sensor networks do not scale well in numbers. In distributed sensor networks, only relevant statistics are shared with neighboring nodes, thus limiting the amount of data as well as the distance over which this data is transmitted.

Thus, distributed networks are ideal for large-scale deployment.

From a sensing point of view, homogeneous sensor networks are a special case of heterogeneous sensor networks in which each of the sensing modalities is identical. From a processing point of view, distributed processing is a generalization of centralized processing since algorithms developed for distributed processing can operate in centralized networks, but the reverse is not necessarily true. The aforementioned reasons make distributed processing in heterogeneous sensor networks a very interesting and challenging research topic that has been explored with limited success in the past. In this dissertation, we focus on the most general case of fully distributed processing in heterogeneous smart sensor networks. However, the methods developed herein are also applicable to distributed sensor networks in which the data from a subset of nodes in the network is processed at common processing units. In the latter case, the processors have access to more data from multiple nodes, thus making it a special case of the former, more basic problem.

We define the *organic state space* for a node as the subspace of the target state space that is observable at that node. A block diagram for a typical smart sensor node is given in Figure 2. The sensor acquires raw data from the environment. This data is fed into the organic pre-processor block, which produces state estimates in the organic state space for that sensor node. Depending on the particular sensor modality, the available processing resources, and the desired target state space, the organic pre-processor could process the sensed data in various ways to produce organic state estimates for that node. This block could perform various activities including, but not limited to, beamforming (for sensor arrays), radar pre-processing, and batch processing of measurements to generate motion estimates. It is important to note that in various situations, the organic state spaces may be dissimilar at the different nodes in the network. Another point to note is that each node may only be able to observe a reduced subspace of the target state space. Such a problem would arise if the target state space is the x-y position of a vehicle and the network consists

of bearing-only sensors and range-only sensors that operate independently. Hence, one-to-many mappings may exist from the various organic state spaces to the target state space. Therefore, it is essential to fuse organic estimates from multiple nodes to come up with reliable estimates in the target state space. Such global estimates can be achieved if the network, as a whole, is capable of observing the target state.

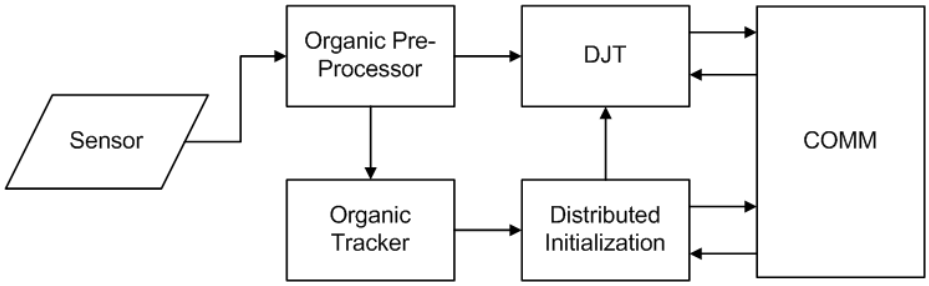


Figure 2. System block diagram of a smart sensor node.

The organic state estimates are used to provide input to the Distributed Joint Tracker (DJT) block that operates in parallel at the different nodes in the network and tracks the system's varying state parameters in the target state space. A detailed discussion of the DJT block is outside the scope of this work. An interested reader is referred to [13]. An organic tracker operates within each node, tracking targets in the organic state space for that node using estimates from the organic pre-processor. Though the organic tracker may seem redundant when the DJT is present, it is actually an essential component since it facilitates the detection of new targets at individual nodes. When a target is detected that does not correspond to existing target tracks in the organic tracker, the organic state estimates for that target are fed into the distributed initialization block. The distributed initialization block takes in organic state estimates for new targets from multiple nodes and combines them to produce the initial state estimates in the target state space used by the DJT. This dissertation will focus on the distributed initialization block. Some of the ideas developed for the initialization algorithm can be extended to the DJT, but such extensions will not be discussed here.

1.2 The Stochastic Filtering Problem

The problem of estimating the posterior distribution of evolving hidden parameters that represent the state of a system, based on a sequence of received measurements that contain partial information about the hidden parameters and a state evolution model, is called the stochastic filtering problem. In general, the evolution of the hidden system parameters is modeled as a Markov process and useful measurements are available in the form of a known function of the unknown parameters.

Let $\mathbf{s}_t$ be the hidden state vector of the system of interest and $\mathbf{z}_t$ be the measurement vector at time t. The state evolution model and the measurement model are given by

$$\mathbf{s}_t = b\left(\mathbf{s}_{t-1}\right) + \mathbf{u}_t, \tag{1}$$

$$\mathbf{z}_t = g\left(\mathbf{s}_t\right) + \mathbf{v}_t, \tag{2}$$

where $\mathbf{u}_t$ and $\mathbf{v}_t$ are the possibly non-Gaussian state and measurement noise vectors respectively, and $b\left(\cdot\right)$ and $g\left(\cdot\right)$ are possibly nonlinear vector valued functions. The goal is to estimate the posterior distribution $p\left(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{z}_t\right)$. Under specific conditions, the optimal solution to the stochastic filtering problem can be determined analytically. For example, if the state space models are linear and Gaussian, then optimum estimates can be acquired using the Kalman filter [14]. However, in most real world applications, linearity and Gaussianity assumptions do not necessarily hold true. In such cases, closed-form solutions to the problem may not exist and analytic evaluation may be impossible. Over the past few decades, various methods for obtaining suboptimal solutions to the filtering problem have been proposed. Some methods, such as the extended Kalman filter (EKF) [15], rely on analytic approximations to locally linearize the state space models. Such approximations can lead to biased estimates of the target distribution [16]. Simulation-based methods, also known as Monte Carlo methods, on the other hand, are not restricted by linearity or Gaussianity assumptions. These methods produce a discrete approximation to the probability distribution of interest by evaluating the target posterior distribution at N randomly

selected points, called *particles*, in the parameter state space. The estimates given by the Monte Carlo methods have been proven to converge as N approaches infinity [7].

The stochastic filtering problem can be extended to sensor network initialization when the state distribution of the hidden parameters of the underlying system needs to be estimated from sequences of measurements acquired by the various sensor nodes in the network, without any prior information about the hidden parameters. In this case, analytical solutions become extremely complicated, especially if processing is distributed among the various nodes in the network. This makes Monte Carlo methods much more attractive than analytic methods in sensor network initialization. However, care must be taken when using Monte Carlo methods in distributed sensor networks since an increase in N corresponds to increases in computation and, depending on the specific implementation, increases in communication. Both processes consume power, which is a scarce resource in battery-powered networks.

1.3 Importance Sampling

The classical importance sampling method of Hammersley and Handscomb [10] is best known as a variance reduction mechanism for evaluating integrals of functions using Monte Carlo methods [17]. In [18], it is shown that importance sampling can reduce the number of samples necessary to obtain the desired nonparametric confidence intervals by 90%-95%, as originally developed by Efron [19]. Hence, importance sampling finds applicability in sensor networks using Monte Carlo methods for data fusion since it provides the network with the ability to reduce the communication bandwidth by effectively reducing the number of samples used to represent messages along the lines of [17, 19, 7].

The idea of importance sampling is the following. The goal is to evaluate the expectation integral,

$$I = E_p\{f\} = \int f(x)p(x)dx, \tag{3}$$

where $p(\cdot)$ denotes a probability density function. For various reasons, analytical evaluation of (3) may be difficult, if not impossible. Instead, Monte Carlo integration methods use N independent samples, or *particles*, $x^{(i)} \sim p(x)$, that lie in highly probable regions of $p(x)$, and obtain an estimate $\hat{I}_p = \sum f(x^{(i)})/N$ [20, 21, 17]. If it is difficult to sample $x^{(i)} \sim p(x)$ but $p(x)$ can be evaluated pointwise, we could sample N independent particles from another density, $x^{(i)} \sim q(x)$, and use $\hat{I}_q = \sum\{f(x^{(i)})p(x^{(i)})\}/\{Nq(x^{(i)})\}$ [10] as our estimator. The second method is called *importance sampling*. The ratio $w^{(i)} = p(x^{(i)})/q(x^{(i)})$ is called the *importance weight*, or simply *weight*, of the particle $x^{(i)}$, and it takes care of any discrepancies resulting from the particle generation with respect to $q(\cdot)$, which is referred to in the literature as the *importance function*.

For importance sampling to provide reliable estimates, the importance function must have a larger region of support than the target distribution $p(x)$. The region of support must not be too large as a majority of the sampled particles would then lie in low probability areas of the target distribution. A large amount of computation would be wasted on such particles that end up contributing minimally to the final estimate. On the other hand, if the region of support of the importance function is much smaller than that of the target distribution, then estimates would be biased and would lie within the high probability region of the importance function. Hence, care must be taken when selecting an importance function since the accuracy of the estimates and the computational load are both directly affected by this choice.

The optimal importance function that reduces the variance of the importance weights is the target posterior distribution itself [7]. In most cases, however, this distribution may not be known, and even if it were known, sampling from it could be extremely difficult, if not impossible. There are various methods for selecting alternative importance functions that are simpler to sample particles from. Here, we give an example in the context of target tracking. Let $\mathbf{z}_{m,t}$ denote the m^{th} node's measurement, or state estimate, at time t. If prior information about the state vector and the current measurement vector are available,

approximations to the target posterior can be made using Bayes' rule since

$$p\left(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{z}_{m,t}\right) \propto p\left(\mathbf{z}_{m,t}|\mathbf{s}_t\right) p\left(\mathbf{s}_t|\mathbf{s}_{t-1}\right), \tag{4}$$

where $p\left(\mathbf{z}_{m,t}|\mathbf{s}_t\right)$ and $p\left(\mathbf{s}_t|\mathbf{s}_{t-1}\right)$ are the data likelihood and state transition probability, respectively. The functional forms of $p\left(\mathbf{z}_{m,t}|\mathbf{s}_t\right)$ and $p\left(\mathbf{s}_t|\mathbf{s}_{t-1}\right)$ might be nonlinear but they are known from the model. In [7], the optimal importance function is approximated as a Gaussian pdf by evaluating the second-order Taylor series expansion of the logarithm of the product density in (4). This choice for the importance function is effective since particles are proposed based on state and measurement information. However, making this Gaussian approximation to the posterior distribution requires global knowledge of all available measurements. This may be possible to implement in centralized sensor networks but may not be possible in distributed sensor networks since individual nodes may not be aware of global measurements.

A simpler importance function that is commonly used in tracking applications is the state transition probability, $p\left(\mathbf{s}_t|\mathbf{s}_{t-1}\right)$. The use of this importance function leads to the commonly used bootstrap filter [22]. Since global knowledge of measurements is not required and because its implementation is simple, the bootstrap filter is commonly used in distributed implementations of the particle filter [13]. However, since this importance function does not consider measurements when proposing particles, algorithms employing this importance function show poor performance when the received measurements disagree with the state transition model. This importance function, though popular in tracking scenarios, is not useful for initialization problems where prior state information is not available.

A good importance function for use in distributed sensor network initialization would make use of measurements from all the nodes when proposing particles without sharing raw data between individual nodes. Our approach uses an importance function consisting of multiple components that can be sampled independently at the different nodes in the network and the particles thus generated are combined together to represent samples from the selected importance function.

1.4 Kernel Density Estimation

Since the algorithms presented in this dissertation are based on Monte Carlo methods, the posterior distributions of interest are represented by particles, and possibly their associated importance weights, resulting in a pmf. This is true even if the target state variables being estimated are in fact continuous. The construction of the underlying pdf from the observed data is called *density estimation*. For our purposes, the observed data consists of particles and weights. Depending on the application, there could be various reasons why one would be interested in estimating the underlying pdf of the state variables of interest, given only the pmf represented by the particles and importance weights. Some of these reasons will become clear when discussed in the following chapters.

Density estimation methods can be broadly classified into two main types: (i) parametric methods and (ii) nonparametric methods. Parametric methods make strong assumptions about the general structure of the underlying pdf and use the observed data to determine parameters that define the exact shape of the assumed pdf. For example, one could assume the state variables are distributed according a multivariate Gaussian distribution. In that case, the mean μ and covariance Σ of the observed data would be calculated and the underlying distribution would be assumed to be $\mathcal{N}(\mu, \Sigma)$. Such parametric methods show satisfactory performance when prior knowledge about the general structure of the underlying pdf is available and it is simple enough to be represented by a standard distribution. If the assumption about the general structure of the underlying pdf is incorrect, then the estimated density function might in no way resemble the true underlying distribution. Nonparametric methods, on the other hand, make no strong assumptions about the underlying distribution. The estimate of the underlying pdf is generated using only the observed data. Since the data is allowed to speak for itself, nonparametric methods show good performance when no prior information about the underlying pdf is available. The first known use of nonparametric density estimation was targeted towards discriminant analysis [23]. Since then, various methods for nonparametric density estimation have been explored in great detail,

a good overview of which can be found in [24]. It is important to note that nonparametric density estimation methods typically have significantly higher computational demands as compared to parametric methods .

In this dissertation, we focus on a particular nonparametric density estimation method known as kernel density estimation (KDE). Assume $p(\cdot)$ represents the underlying d-dimensional pdf, and $\{\mathbf{x}^{(i)}\}_{i=1}^{D}$ represents the D observed data points. Using KDE, the estimated pdf $\hat{p}(\cdot)$ is given by

$$\hat{p}(\mathbf{x}) = \frac{1}{\alpha} \sum_{i=1}^{D} K\left(\frac{\left(\mathbf{x} - \mathbf{x}^{(i)}\right)}{h}\right), \tag{5}$$

where h is the bandwidth, or smoothing parameter, of the kernel function K, and α is a normalization constant. One can view (5) as the sum of bumps located at the D observations, each bump defined by the choice of kernel and bandwidth. It has been shown in [24] that if $K(\cdot)$ is itself a pdf, then the density estimate obtained using (5) is also a pdf. Additionally, the estimated density inherits all the continuity and differentiability properties of $K(\cdot)$.

Many possible kernel functions exist, and the particular choice of the kernel can affect the quality of the final density estimate. However, it has been shown in [24] that the specific choice of K is not as important as the correct choice of the kernel bandwidth. As the bandwidth $h \to 0$, the estimated pdf reduces to a sum of Dirac delta functions. On the other hand, if $h \to \infty$, all the detail in the estimated pdf is lost. Various methods for choosing the bandwidth are discussed in [24].

For most of this dissertation, we use a simplified notation,

$$\hat{p}(\mathbf{x}) = \sum_{i=1}^{D} W\left(\mathbf{x} - \mathbf{x}^{(i)}\right), \tag{6}$$

where the effect of the smoothing parameter h, kernel shape K, and scaling $\frac{1}{\alpha}$, are all incorporated into the single term $W(\cdot)$. If importance sampling is used, as is the case in the majority of this dissertation, additional information about the particles is contained in the associated D importance weights $\{w^{(i)}\}_{i=1}^{D}$. To incorporate the effect of the importance

weights, (6) is modified as

$$\hat{p}(\mathbf{x}) = \sum_{i=1}^{D} w^{(i)} W\left(\mathbf{x} - \mathbf{x}^{(i)}\right). \tag{7}$$

If the importance weights sum to unity, which is the case after normalization in most applications of importance sampling, (7) represents a pdf and it inherits the continuity and differentiability properties of $K(\cdot)$.

1.5 Weighting and Resampling

In applications using Monte Carlo methods, particularly those using importance sampling, the degree of belief in a particular particle is represented by the importance weight assigned to that particle. In Markov chain Monte Carlo methods, or particle filters, that track evolving target states in time, these importance weights are also used for *resampling*, a method used to inhibit degeneracy. Resampling typically involves a weighted sampling with replacement operation in which the resampling weight assigned to each particle is identical to the importance weight for that particle. Thus, when used with particle filters, resampling shifts the particle distribution towards high probability areas of the state space, eliminating particles with low importance weights and replicating particles with high importance weights.

In this dissertation, resampling is implemented in a different manner compared to the traditional implementation used by particle filters. Instead of trying to move particles to high probability regions of the state space, we use resampling for two main purposes: (i) to combine individual density components into a mixture, ensuring that the resulting mixture density has all the individual components equally weighted and (ii) to maintain fixed bandwidth for inter-node communication. Because the individual component densities are represented by particles and importance weights, we use a different set of weights, which we call *resampling weights*, to drive the resampling operation. The resampling weights should not be confused with the importance weights.

Assume we have two incoming densities,

$$p_1(\mathbf{s}) = \frac{1}{M_1} \sum_{m=1}^{M_1} p_{1,m}(\mathbf{s}), \tag{8}$$

$$p_2(\mathbf{s}) = \frac{1}{M_2} \sum_{m=1}^{M_2} p_{2,m}(\mathbf{s}), \tag{9}$$

each representing an equally weighted mixture of individual component densities. Note that M_1 and M_2 need not be equal. We wish to combine these two mixture densities into a single mixture that is equally weighted in all the individual components. Thus, the desired combined mixture density can be represented as

$$p_D(\mathbf{s}) = \frac{1}{M_1 + M_2} \left(\sum_{m=1}^{M_1} p_{1,m}(\mathbf{s}) + \sum_{m=1}^{M_2} p_{2,m}(\mathbf{s}) \right). \tag{10}$$

From 8, 9 and 10, it is clear that without explicit knowledge of the individual component densities, $p_D(\mathbf{s})$ can be generated from $p_1(\mathbf{s})$ and $p_2(\mathbf{s})$ by taking

$$p_D(\mathbf{s}) = \check{w}_1 p_1(\mathbf{s}) + \check{w}_2 p_2(\mathbf{s}), \tag{11}$$

where the resampling weights $\check{w}$ are given by

$$\check{w}_1 = \frac{M_1}{M_1 + M_2}, \tag{12}$$

$$\check{w}_2 = \frac{M_2}{M_1 + M_2}. \tag{13}$$

The only information needed is the number of component densities contained within each incoming mixture density.

In our work, each incoming mixture density is represented by D particles and importance weights. We wish to combine the incoming mixture densities into a single equally weighted mixture represented by D particles and weights. The idea for resampling explained above can be easily extended to such discrete representations. Just as in the continuous case, resampling weights are generated based on the number of individual component densities present in the incoming mixture densities. However, instead of assigning a resampling weight directly to a density, it is assigned to each particle representing that density.

For the general case in which we have N incoming mixture densities, we have a total of ND particles, each of which is assigned a resampling weight. A weighted sampling with replacement operation can be used to generate a set of D particles. The importance weights associated with the surviving particles are stored. The surviving particles, along with their importance weights, represent the desired combined mixture distribution.

Let us now illustrate the resampling algorithm with a simple example shown in Figure 3. Two one-dimensional incoming mixture densities are available, each represented by $D = 10000$ particles. In this example, we assume that the importance weights are all equal and are therefore dropped from our consideration. Thus, the densities are simply represented by the particle distribution. Mixture 1 is an equally weighted mixture of 3 Gaussian densities, whereas mixture 2 is an equally weighted mixture of 2 Gaussian densities. Since both mixtures use the same total number of particles, the number of particles representing a single component density in mixture 2 is greater than that in mixture 1. This can clearly be seen in the upper histograms of Figure 3. Following the theory given above, the resampling weights for the particles in mixtures 1 and 2 and given by $\check{w}_1 = \frac{3}{5}$ and $\check{w}_2 = \frac{2}{5}$ respectively. Note that for weighted resampling, only relative weighting matters. Therefore we assign resampling weights as $\check{w}_1 = 3$ and $\check{w}_2 = 2$. After a weighted resampling with replacement operation on the total set of $2D$ incoming particles according to the associated resampling weights, a single combined mixture density represented by D particles is generated. It can be seen in the lower histogram of Figure 3 that each component density is equally weighted in the final combined mixture density. This example illustrates the basic idea of our resampling algorithm, which is a critical component of our initialization algorithm for fusing data at each node. The resampling operation is relatively straightforward when the individual component densities in the various incoming mixtures are unique. Care must be taken when common component densities are present in multiple incoming mixtures to ensure that they are not over represented in the final combined mixture density. Such an implementation can be found in Chapter 3.

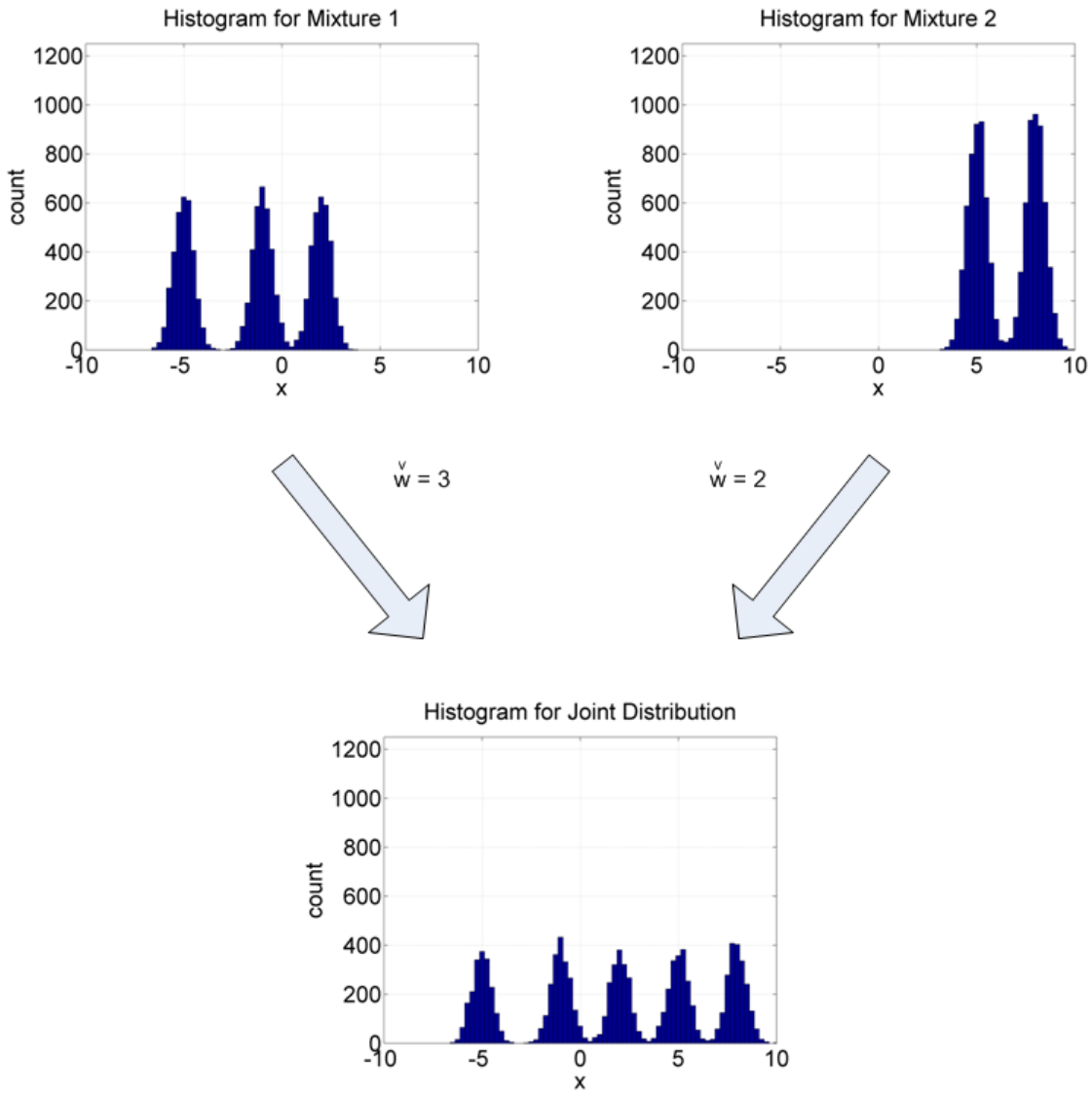


Figure 3. Resampling operation. The upper plots show histograms for the two incoming mixture densities. The lower plot shows the histogram for the combined mixture density.

CHAPTER 2

INITIALIZING SENSOR NETWORKS USING COMMUNICATION AND COMPUTATION CHAINS

2.1 Introduction

Since the focus of this dissertation is the initialization of a target state parameter vector in distributed heterogeneous sensor networks, this chapter develops the initialization algorithm for a very simple communication topology. The discussion herein is limited to sensor networks in which sensor nodes communicate with each other using a fixed, one-hop communication chain, as given in Figure 4.

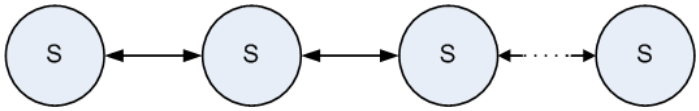


Figure 4. Fixed one-hop communication chain. S represents a sensor node and the double headed arrows represent two way communication.

Limiting the focus to such a simple communication topology allows us to divorce the communication problem from the distributed data fusion problem. Therefore, we can develop our initialization algorithm for a system in which data collected from the various nodes is used in a sequential manner, and worry about modifying the algorithm for use with more general communication topologies in later chapters.

We first develop a low complexity initialization algorithm that requires three communication passes through the network to achieve global initialization. Since this is the simplest form of the initialization algorithm and forms the basis for the more specialized and complicated algorithms described in this dissertation, it will be developed in far greater detail than the other versions of the algorithms that follow. Next, a low latency version of the initialization algorithm is developed. In this version, at the cost of an increase in intra-node computation, global initialization is achieved in only two communication passes through the network. Finally, the low complexity algorithm is modified to compensate for various

delays that may enter the system, in particular, delays arising from processing and communication latencies, as well as finite signal propagation velocities. In all three algorithms, the final communication pass is only required to broadcast the final estimates throughout the network and requires no additional computation. To demonstrate applicability in a real world system, the algorithms are implemented in a surveillance scenario using a sensor network consisting of direction of arrival (DOA) nodes and range-Doppler nodes. The goal is to produce initial estimates of multiple targets' positions and velocities in a fully distributed manner. The algorithm is tested on simulated data and the resulting state vector distributions are displayed.

2.2 Low Complexity Initialization Algorithm

In this section, we describe the low complexity Monte Carlo algorithm used to generate initial probability distributions of the system's state parameter vector $\mathbf{s}_t$. Since prior state information is not available, the initial probability distribution must be generated using only the organic estimates available at the assorted heterogeneous sensor nodes. Let M be the total number of nodes in the network and let $\mathbf{z}_t$ be the set of organic state estimates from all M nodes. The goal is to generate a discrete approximation to the posterior distribution of the global state in the network, $p(\mathbf{s}_t|\mathbf{z}_t)$, represented by D particles, $\{\mathbf{s}_t^{(i)}\}_{i=1}^{D}$, and possibly their associated importance weights, $\{w_t^{(i)}\}_{i=1}^{D}$. For simplicity, we assume a fixed one-hop communication path from the first node to the last node in the network. The pseudocode for the implementation of this algorithm is given in Appendix A. We begin by developing the algorithm theoretically and discuss issues relating to communication and computation. Next, we go over the step-by-step implementation of the algorithm in a real world application.

2.2.1 Theoretical Development

Using the Monte Carlo methodology to produce a discrete approximation to the initial distribution of the posterior, one should sample particles from the posterior distribution

itself to obtain the optimal particle distribution [7]. Analytically representing the posterior requires the simultaneous knowledge of measurements from all the nodes in the network, a requirement that may not be realistic in distributed networks. However, one can make use of importance sampling theory to sample particles from a well selected importance function and eliminate biases by assigning appropriate importance weights to the particles.

Using Bayes' rule, the posterior distribution can be expressed as

$$p(\mathbf{s}_t|\mathbf{z}_t) = \frac{p(\mathbf{z}_t|\mathbf{s}_t)p(\mathbf{s}_t)}{p(\mathbf{z}_t)}. \tag{14}$$

Given the state of the system, the measurements, and hence the organic state estimates, available at the different nodes can be assumed to be independent. Hence, the sensor network data likelihood can be factored into the product of the individual node data likelihoods in the network. Since this is an initialization problem, prior information about the target state is not available. Therefore, $p(\mathbf{s}_t)$ is chosen to be a non-informative uniform distribution and is dropped from the equation. Note that $p(\mathbf{z}_t)$ is simply a proportionality constant that is independent of the state. Hence, (14) is simplified as

$$p(\mathbf{s}_t|\mathbf{z}_t) \propto \prod_{m=1}^{M} p(\mathbf{z}_{m,t}|\mathbf{s}_t) \propto \prod_{m=1}^{M} p(\mathbf{s}_t|\mathbf{z}_{m,t}), \tag{15}$$

2.2.1.1 Choice of the Importance Function

It can be clearly seen in (14) and (15) that analytically representing the posterior requires knowledge of the organic state estimates from all the nodes in the network. If we were to sample particles directly from the posterior distribution, then (*i*) organic estimates from all the different nodes must first be collected, (*ii*) the posterior distribution must be evaluated analytically, and (*iii*) particles must be sampled from the posterior. Condition (*i*) makes efficient distributed implementation difficult, whereas conditions (*ii*) and (*iii*) may be impossible to implement. We look for an importance function that satisfies the following criteria: (*a*) the particle support can be efficiently generated by sampling the importance function sequentially at the different nodes in the network, (*b*) the communication bandwidth should be limited to a predetermined amount regardless of the number of nodes in

the network, (*c*) all nodes that have provided input to the initialization algorithm should be weighted equally and (*d*) only those nodes that have a detection should be used to generate particles and weights. Here, condition (*a*) ensures a fully distributed and efficient implementation, condition (*b*) ensures scalability, condition (*c*) ensures that the output will remain unchanged regardless of the order in which the different nodes provide input to the algorithm, and condition (*d*) saves particles from being wasted on nodes that detect nothing. A simple choice for such an importance function is an equally weighted mixture of the individual posterior distributions from the different nodes,

$$\pi(\mathbf{s}_t|\mathbf{z}_t) = \frac{1}{M}\sum_{m=1}^{M} p(\mathbf{s}_t|\mathbf{z}_{m,t}). \tag{16}$$

Sampling this importance function can be accomplished by sampling the individual posterior distributions at the different nodes and then combining the sets of samples to generate the final set of particles. In other words, we can sample from this importance function in a distributed, sequential manner without sharing raw data between nodes, as elaborated in the following sections.

2.2.1.2 Observability

We first show that it is possible to sample particles from the individual posterior distributions at the different nodes in the network. Let the state vector $\mathbf{s}_t$ be n-dimensional and let the organic estimates at node m be random realizations of s-dimensional feature vectors $\hat{\mathbf{z}}_{m,t}$. We can assume that each feature is a function of the state vector since those features that are not functions of the state vector do not provide any information to determine the posterior of interest and hence can be discarded. Thus the feature vector at node m is given by

$$\hat{\mathbf{z}}_{m,t} = f_m(\mathbf{s}_t) = \begin{bmatrix} f_{m,1}(\mathbf{s}_t) \\ f_{m,2}(\mathbf{s}_t) \\ \vdots \\ f_{m,s}(\mathbf{s}_t) \end{bmatrix}. \tag{17}$$

We now determine the conditions that $f_m(\cdot)$ must satisfy to enable one to sample from (16). Let $f_m(\cdot)$ be a continuously differentiable vector valued function. If and only if all features in the feature vector at a particular node provide complementary information without redundancy, is $\nabla f_m(\mathbf{s}_t)$ nonsingular and

$$\det(\nabla f_m(\mathbf{s}_t)) \neq 0. \tag{18}$$

If any of the features provide redundant information, those particular features can be discarded to give a feature space of reduced dimension and no redundancy. Therefore, we can assume all features provide complementary information and (18) is satisfied.

First, consider the case when $s < n$. Given the features $\hat{\mathbf{z}}_{m,t}$, the system in (17) is underdetermined. Therefore, there exist infinitely many solutions for $\mathbf{s}_t$. These solutions form a level set in the target state space. Because real world sensors have finite operating range, the level set of solutions can be constrained to have finite dynamic range. In some cases, the level set can be represented by explicit equations relating the state variables. However, this may not be possible in most cases even though the level sets do exist. Let α be any solution of (17). By the implicit function theorem [25], in the neighborhood of α, the level set $L_f(\hat{\mathbf{z}}_{m,t})$ is an $n - s$ dimensional manifold. Let Λ represent the set of all such manifolds. Particles can be generated by sampling uniformly from points in Λ since each of these points is a possible solution to (17).

Now consider the case when $s = n$. By the inverse function theorem [25], given features $\hat{\mathbf{z}}_{m,t}$, a unique inverse function $f_m^{-1}(\cdot)$ exists in the neighborhood of $\hat{\mathbf{z}}_{m,t}$ and therefore there exists a unique solution to (17) given by

$$\mathbf{s}_t = f_m^{-1}(\hat{\mathbf{z}}_{m,t}). \tag{19}$$

Thus the target state can be uniquely determined.

Since the organic estimates $\mathbf{z}_{m,t}$ are derived from noisy sensor data, they can be considered to be random realizations of the feature vectors. We propose to use the organic

estimates at the various nodes as estimates of the true feature vectors in the implementation of the preceding procedure. To account for estimation errors, the proposed particles are randomly perturbed based on the variance determined by the measurement model. In this manner, particles can be sampled from the individual posterior distributions without sharing raw data.

If node m is a binary detection sensor (i.e., the sensor's output depends only on whether a phenomenon of interest is detected or not) then the mapping from the target state space to the feature space is not differentiable. However, this is a special case since the output is a binary function on some $f_m(\cdot)$. In this situation, all points in the domain of $f_m(\cdot)$ that would result in a detection are possible target states and can be denoted by a set S_d. The same rules for sampling explained above can be applied to points in S_d and the procedure remains unchanged.

2.2.1.3 Practicalities of Message Passing

Since the particles are proposed sequentially at the different nodes in the network and then combined together to generate the final particle support, the information propagated between nodes during the particle proposal stage consists of the particles themselves. Say, for example that one desires to represent the posterior using D particles. A simple method to produce samples according to (16) would be to sample D/M particles from each node and simply combine them together to generate the final set of D particles. This method, despite being intuitive, has an inherent drawback. Assume node m does not have a detection. In this case, the D/M particles representing the posterior at node m would be uniformly distributed over the natural state space for that node. These particles do not represent any useful information about the target state. It would make sense not to sample from such nodes that do not have detections but instead to sample particles representing an equally weighted mixture of only those local posteriors that are associated with nodes that have detections. Hence, more particles cover the state space of interest. This can be accomplished by using a weighted resampling operation that ensures that the individual

posteriors for nodes with detections are equally weighted, irrespective of the total number of nodes in the network. The resampling algorithm follows the basic idea discussed in Section 1.5. Assume that node m receives D particles and a single number n_s from node $(m-1)$. These particles represent $\frac{1}{n_s}\sum_{\hat{m}=1}^{m-1} p(\mathbf{s}_t|\mathbf{z}_{\hat{m},t})$, an equally weighted mixture of the posterior distributions from the n_s nodes that detected a target and provided input to the algorithm. Therefore, each of these particles is assigned a resampling weight of n_s. Here we assume that a node that does not detect a target sets its posterior distribution to 0, thus avoiding making a contribution to the initialization algorithm. If node m does not have a detection, then the received particles and n_s are transmitted to node $(m+1)$. However, if node m has a detection, it generates its own set of D new particles by sampling from its local posterior distribution, $p(\mathbf{s}_t|\mathbf{z}_{m,t})$. Since these new particles represent information from the current node only, they are each assigned a resampling weight of 1. We point out once again that these resampling weights should not be confused with the importance weights. The resampling weights are used solely to ensure that (16) can be sampled from sequentially. The importance weights, assigned in the following section, are used along with the final set of particles to represent the desired posterior distribution. From the total set of $2D$ weighted particles, a set of D equally weighted particles is generated by performing a weighted sampling with replacement operation according to the resampling weights. This final set of D particles represents $\frac{1}{n_s+1}\sum_{\hat{m}=1}^{m} p(\mathbf{s}_t|\mathbf{z}_{\hat{m},t})$ and is transmitted to node $(m+1)$ along with a single number, (n_s+1). The resampling operation does not require synchronization of the nodes and ensures that the posterior distributions from nodes with detections are equally weighted in the importance function. At the end of a forward communication pass through the network, the distribution of the final set of particles at node M represents the importance function (16).

2.2.1.4 Assigning Importance Weights

The final set of particles must be distributed throughout the network, and at the same time, importance weights must be computed for these particles so that the final set of particles

and importance weights represents the posterior distribution $p(\mathbf{s}_t|\mathbf{z}_t)$. For scalability reasons, raw data is not shared between nodes. Therefore, the final importance weights must be determined by first evaluating individual components of the importance weights at the various nodes in the network and sequentially updating the importance weights as they propagate through the network.

Using the results of [7], the importance weights for our problem are given by

$$w_t^{(i)} = \frac{p\left(\mathbf{s}_t^{(i)}|\mathbf{z}_t\right)}{\pi\left(\mathbf{s}_t^{(i)}|\mathbf{z}_t\right)}, \tag{20}$$

where $\pi\left(\mathbf{s}_t^{(i)}|\mathbf{z}_t\right)$ is the importance function. Using (15) and (16), (20) can be simplified as

$$w_t^{(i)} \propto \frac{\prod_{m=1}^{M} p\left(\mathbf{s}_t^{(i)}|\mathbf{z}_{m,t}\right)}{\sum_{m=1}^{M} p\left(\mathbf{s}_t^{(i)}|\mathbf{z}_{m,t}\right)} \propto \frac{\prod_{m=1}^{M} p\left(\mathbf{z}_{m,t}|\mathbf{s}_t^{(i)}\right)}{\sum_{m=1}^{M} p\left(\mathbf{s}_t^{(i)}|\mathbf{z}_{m,t}\right)}. \tag{21}$$

Using Bayes' rule, we obtain the following expression for the local posteriors

$$p\left(\mathbf{s}_t^{(i)}|\mathbf{z}_{m,t}\right) = \frac{p\left(\mathbf{z}_{m,t}|\mathbf{s}_t^{(i)}\right) p\left(\mathbf{s}_t^{(i)}\right)}{p\left(\mathbf{z}_{m,t}\right)}. \tag{22}$$

Since no prior information about the state vector is available, $p(\mathbf{s}_t)$ is assumed uniform over its natural space and is dropped from the equation. Thus, (21) simplifies to

$$w_t^{(i)} \propto \frac{\prod_{m=1}^{M} p\left(\mathbf{z}_{m,t}|\mathbf{s}_t^{(i)}\right)}{\sum_{m=1}^{M} \frac{p\left(\mathbf{z}_{m,t}|\mathbf{s}_t^{(i)}\right)}{p\left(\mathbf{z}_{m,t}\right)}}. \tag{23}$$

It can be seen that the importance weights for the particles can be calculated, up to a proportionality constant, by evaluating a quotient in which the numerator is the product of the data likelihoods from all the nodes in the network and the denominator is a weighted sum of the same likelihoods. Hence, a sequential update of the importance weights can be accomplished if the numerators and denominators are both communicated between nodes.

2.2.1.5 Missed Sensor Detections

Given our choice of the weighting function in (23), a robust likelihood function is essential for accurate assignment of the importance weights to different particles. If a simple Gaussian likelihood function is used in the above equations and the likelihood of a particle is

almost zero at one of the nodes, then its overall importance weight will also be close to zero. This would occur if even one of the nodes in the network failed to detect the event of interest. In such situations, one would not want the overall importance weight of the particle to be zero since the target event is present with high probability. To avoid this degeneracy, it is important that a robust likelihood function that accounts for missed detections be used.

The approach used here is similar to the approach used in [6, 26]. Assume that node m produces K organic estimates. Then, given a particle $\mathbf{s}_t^{(i)}$, the estimates $\mathbf{z}_{m,k,t}$, $k = 1, ..., K$, could correspond either to the target event or to clutter. The clutter distribution is assumed to be Poisson with spatial density λ. The probability of miss is set equal to a constant q. It is assumed that there is an equal probability for each of the K organic estimates to correspond to the true target event, and the organic estimate corresponding to the true target is normally distributed about that target's state. Thus, as shown in [6] the likelihood function can be expressed as

$$p\left(\mathbf{z}_{m,t}|\mathbf{s}_t^{(i)}\right) \propto 1 + \frac{1-q}{\sqrt{(2\pi)^s|\Sigma|}q\lambda K} \cdot \sum_{k=1}^{K} \exp\left\{-\frac{1}{2}\left(\mathbf{z}_{m,k,t} - f_m\left(\mathbf{s}_t^{(i)}\right)\right)^{\mathrm{H}} \mathbf{\Sigma}_{m,t}^{-1}\left(\mathbf{z}_{m,k,t} - f_m\left(\mathbf{s}_t^{(i)}\right)\right)\right\}, \tag{24}$$

where s is the dimensionality of the organic state space at node m and $\mathbf{\Sigma}_{m,t}$ is the covariance of the Gaussian distribution. The final set of particles and their associated importance weights gives a discrete representation of the desired posterior distribution.

2.2.1.6 Communication and Computation

The low complexity initialization algorithm requires three passes through the communication chain for global initialization: a forward pass to sequentially generate the particle support, a reverse pass to disseminate the final particles throughout the network and to sequentially generate importance weights, and a forward pass to disseminate the final importance weights throughout the network. Data processing occurs only in the first two communication passes. At each node, the computational load is $O(D)$ operations, where D represents the number of particles.

In the first pass, a fixed number of D particles representing an equally weighted mixture of local posteriors from all preceding nodes is transmitted through the communication chain. Along with the particles, a single number n_s representing the number of nodes that detected a target and provided their input to the algorithm is transmitted. At the end of the first pass, node M has the final set of D particles that represent particles proposed using (16). These particles need to be propagated back to all the other nodes so the importance weights can be computed.

In the second pass, the communication path is reversed. The final set of D particles is propagated back sequentially to node 1. It can be seen in (23) that the individual components of the numerator and denominator of the importance weights can be evaluated independently at each node and can be transmitted cumulatively. Thus the data communicated between pairs of nodes consists of the final set of D particles, the D numerator and D denominator components representing the cumulative importance weights from the preceding nodes. At the end of the second pass, all nodes in the network have the final set of particles and node 1 is the only node with the final set of importance weights.

In the third pass, the final set of D importance weights is propagated throughout the network using the forward communication path. At the end of the third pass, all nodes share the same particles and importance weights and the network is initialized.

2.2.2 Simulation

In this section, we implement the initialization algorithm given in Section 2.2.1 in a surveillance scenario to initialize a multi-target tracker. The tracker is a distributed particle filter based tracker, similar to the one given in [13, 27]. Each particle, $\mathbf{s}_t^{(i)}$, represents a hypothesized target state and the associated importance weight, $w_t^{(i)}$, represents the probability that the particle represents a target's true state. The tracker is implemented using synchronized particle filters that run at each node in the smart sensor network. Synchronized particle filters maintain the same set of particles at each node using synchronized noise sources or noise tables. In [13], the importance function is simply the state transition distribution,

$p(\mathbf{s}_t|\mathbf{s}_{t-1})$. For this choice of the importance function, the particle weights are proportional to the data likelihood, $p\left(\mathbf{z}_t|\mathbf{s}_t^{(i)}\right)$, with respect to that particle. By exploiting the conditional independence of the measurements given the targets' states, the overall data likelihood can be factored into the product of the individual data likelihoods at the M nodes:

$$p(\mathbf{z}_t|\mathbf{s}_t) = \prod_{m=1}^{M} p(\mathbf{z}_{m,t}|\mathbf{s}_t). \tag{25}$$

The product in (25) can be evaluated in a sequential manner with each node providing its local input to the overall importance weights. In this section, the initialization algorithm given in Section 2.2.1 is used to generate the initial set of particles and importance weights for the synchronized particle filters that is common at all the nodes in the network.

The goal is to generate probability distributions representing multiple targets' states in the $\mathbf{s}_t = [x\ y\ v_x\ v_y]^{\mathrm{T}}$ space, where x and y are the Cartesian coordinates of a target's location, and v_x and v_y are the velocity components along the x-y directions. The two types of sensor nodes used to demonstrate the proposed algorithm are DOA nodes (e.g., acoustic arrays, seismic arrays, video cameras, etc.) and range-Doppler nodes (e.g., radar, acoustic motes), since these sensing modalities are representative of the most commonly used sensors for localization and tracking. Note that none of these nodes alone can observe the state space of interest. In fact, if only range and bearing estimates were available at the various nodes, the network as a whole would still be unable to observe the velocity components of the desired state vector $\mathbf{s}_t$ without additional pre-processing of the raw measurements. Hence, some level of organic pre-processing is required at a subset of the nodes in the network so that the network as a whole can observe $\mathbf{s}_t$. However, no node is required to be able to observe the target state vector by itself.

Due to reduced subspace observability, the organic state spaces at the various nodes may have lower dimensionality than $\mathbf{s}_t$, leading to non-unique mappings from fixed points in the organic state spaces to $\mathbf{s}_t$. Each sensor node also runs an independent organic tracker that can track targets in the organic state space at that node. Detailed descriptions about such DOA and range-Doppler trackers can be found in [28, 29, 30, 31, 32, 33, 34, 35, 36, 37]. By

processing batches of DOA estimates in the organic pre-processor, a DOA sensor node can estimate the target's bearing θ, heading direction ϕ, and the natural logarithm of the ratio of the speed to the range Q. Batch processing can also provide error estimates for these parameters. Thus the organic DOA trackers operate in the $[\theta\ Q\ \phi]^{\mathrm{T}}$ space. The organic range-Doppler trackers operate in the $[r\ v_r]^{\mathrm{T}}$ space, where r is the range to the target and v_r is the target's radial velocity. These estimates are easily obtained using radar returns and Doppler shift, and the error can be estimated in advance.

It is important to note that the proposed initialization algorithm produces accurate estimates of the posterior distribution even if simpler, lower dimensional organic state spaces are used. The only condition for the initialization algorithm to produce accurate results is that the nodes in the network should collectively be able to observe the target's full parameter state vector. However, in the remainder of this section, we focus on the $[\theta\ Q\ \phi]^{\mathrm{T}}$ organic state space for DOA nodes and the $[r\ v_r]^{\mathrm{T}}$ state space for range-Doppler nodes. The sensor network is assumed to be calibrated so that each node is aware of its own absolute location and orientation. However, nodes need not be aware of the locations or orientations of other nodes in the network.

2.2.2.1 Sampling Particles

Using the sequential sampling algorithm given in Section 2.2.1, particles can be sampled from the individual posterior distributions from the DOA and range-Doppler nodes as given in Table 1.

Here, $(s_{m,x}, s_{m,y})$ represents the x-y position of node m. Even after pre-processing batches of θ estimates at the DOA nodes to produce estimates of Q and ϕ, the targets' ranges are not known locally. If the targets' ranges were known, then combining them with the organic state estimates would give unique mappings to $\mathbf{s}_t$. Therefore, the range estimate for each particle must be sampled from an uninformative distribution that can be bounded above by the maximum detection range for that sensor. The uniform distribution is an ideal candidate from which to sample random range values to model the range uncertainty.

Table 1. Sampling particles in the target state space from lower dimensional organic estimates. Step I generates particles in the organic state space and the unobservable dimensions are sampled uniformly over their dynamic range. Step II maps the particles generated in step I to the target state space.

	DOA Nodes	**Range-Doppler Nodes**
Step I	$r^{(i)} \sim U\left[0, r_{\max}\right)$	$r^{(i)} \sim \mathcal{N}\left(r_{m,t}, \sigma_{r_{m,t}}\right)$
	$\theta^{(i)} \sim \mathcal{N}\left(\theta_{m,t}, \sigma_{\theta_{m,t}}\right)$	$\theta^{(i)} \sim U\left[0, 2\pi\right)$
	$Q^{(i)} \sim \mathcal{N}\left(Q_{m,t}, \sigma_{Q_{m,t}}\right)$	$v_r^{(i)} \sim \mathcal{N}\left(v_{r_{m,t}}, \sigma_{v_{r_{m,t}}}\right)$
	$\phi^{(i)} \sim \mathcal{N}\left(\phi_{m,t}, \sigma_{\phi_{m,t}}\right)$	$v_t^{(i)} \sim U\left(-\sqrt{v_{\max}^2 - \left(v_r^{(i)}\right)^2}, \sqrt{v_{\max}^2 - \left(v_r^{(i)}\right)^2}\right)$
Step II	$x_t^{(i)} = r^{(i)} \cos\left(\theta^{(i)}\right) + s_{m,x}$	$x_t^{(i)} = r^{(i)} \cos\left(\theta^{(i)}\right) + s_{m,x}$
	$y_t^{(i)} = r^{(i)} \sin\left(\theta^{(i)}\right) + s_{m,y}$	$y_t^{(i)} = r^{(i)} \sin\left(\theta^{(i)}\right) + s_{m,y}$
	$v_{x_t}^{(i)} = e^{Q^{(i)}} r^{(i)} \cos(\phi^{(i)})$	$v_{x_t}^{(i)} = v_r^{(i)} \cos\left(\theta^{(i)}\right) + v_t^{(i)} \sin\left(\theta^{(i)}\right)$
	$v_{y_t}^{(i)} = e^{Q^{(i)}} r^{(i)} \sin\left(\phi^{(i)}\right)$	$v_{y_t}^{(i)} = v_r^{(i)} \sin\left(\theta^{(i)}\right) - v_t^{(i)} \cos\left(\theta^{(i)}\right)$

When these ranges are combined with $[\theta\ Q\ \phi]^{\mathrm{T}}$ estimates available at the DOA nodes, the state vector $\mathbf{s}_t^{(i)}$ for each particle is uniquely determined and particles are spread over the possible solution space. Similarly, for range-Doppler nodes operating in the $[r\ v_r]^{\mathrm{T}}$ state space, tangential velocity and bearing information is required to produce unique mappings to $\mathbf{s}_t$. In applications in which the radar sensors use narrow beams, the targets' bearings may be observable. Similar to the DOA nodes, performing batch processing on these bearings can generate estimates of Q and ϕ, which when combined with the available range and radial velocity estimates can generate estimates of tangential velocity. However, using narrow beams reduces the coverage area of the radar sensors. In this work, we focus on radar

sensors with hemispherical coverage. Such radar sensors are unable to observe targets' bearings and tangential velocities and are more general than sensors with limited angular coverage. In this case, the tangential velocity for each particle can be sampled from an appropriate uniform distribution that takes into account the maximum possible speed of the target. The bearing for each particle can be sampled uniformly over the angular coverage for that node, which in the case of radar sensors with hemispherical coverage, would be between 0 and 2π radians.

2.2.2.2 Initialization Example

Assume that two targets appear simultaneously in the network, with initial states given by $\mathbf{s}_1 = [-200,\ -500,\ 10,\ 20]^{\mathrm{T}}$ and $\mathbf{s}_2 = [1600,\ 0,\ -14,\ -14]^{\mathrm{T}}$. There are a total of four sensor nodes in the field: two bearing nodes located at (500m, 400m) and (800m, −300m), and two range-Doppler nodes located at (200m, −200m) and (1200m, 200m). The node and target positions are shown in Figure 5. Organic trackers at the four nodes detect these targets and produce estimates in their own state spaces.

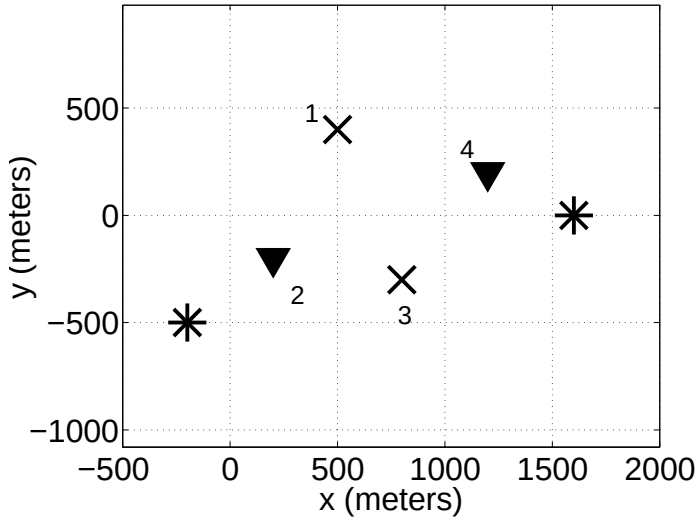
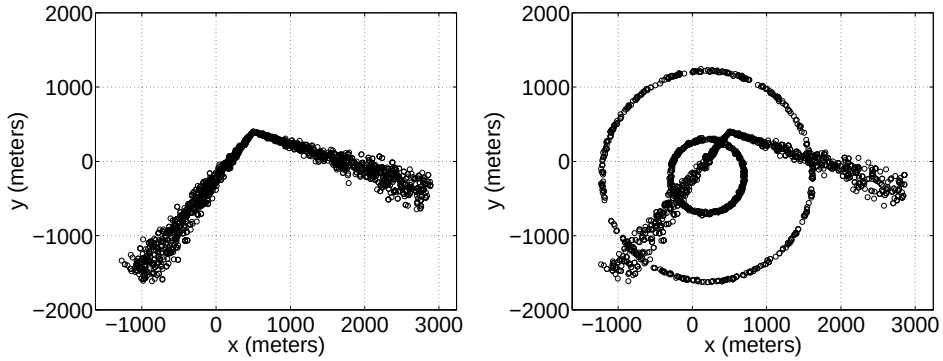


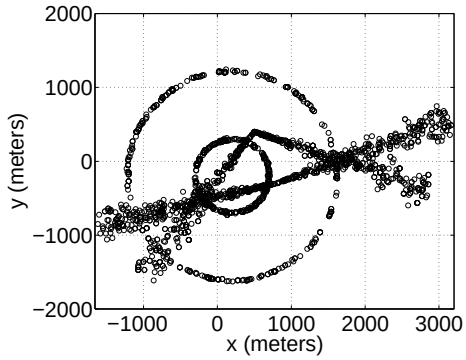
Figure 5. Node and target positions: ∇ represent range-Doppler nodes, X represent bearing nodes and * represent targets.

To simulate the estimates available from the organic trackers (i.e., $[\theta\ Q\ \phi]^{\mathrm{T}}$ from the DOA trackers and $[r\ v_r]^{\mathrm{T}}$ from the range-Doppler trackers), the organic estimates provided

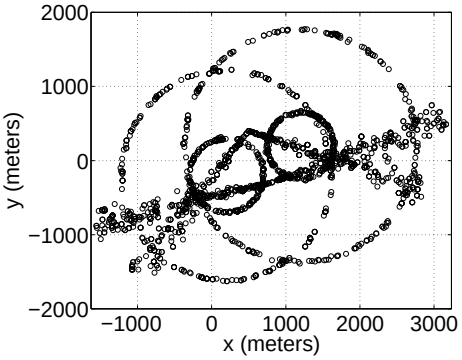


(a) Sensor 1

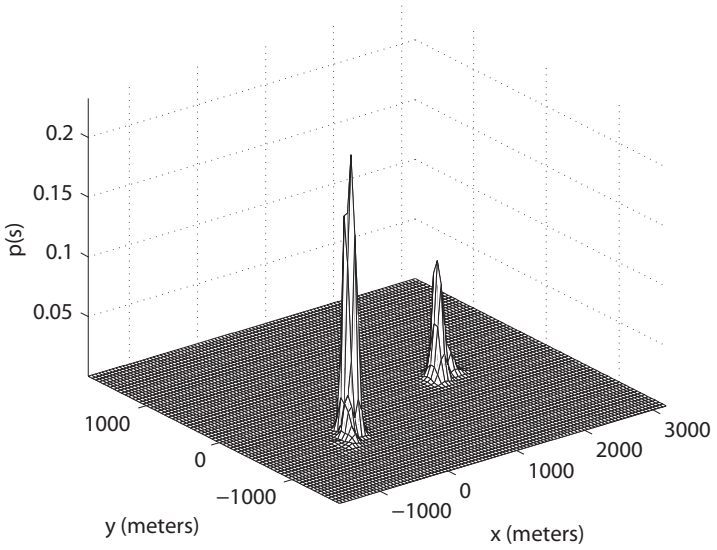
(b) Sensors 1,2



(c) Sensors 1,2,3



(d) Sensors 1,2,3,4



(e) Posterior distribution

Figure 6. Simulation example for initializing multiple targets using the low complexity initialization algorithm.

to the initialization algorithm by the different nodes are Gaussian distributed about their

true values, with standard deviations given by

$$\sigma_\theta = 2^\circ,\ \sigma_Q = 0.02\ \mathrm{s}^{-1},\ \sigma_\phi = 8^\circ, \tag{26}$$

$$\sigma_r = 6\ \mathrm{m},\ \sigma_{v_r} = 0.4\ \mathrm{m/s}. \tag{27}$$

The Gaussianity assumption is made only to simplify the implementation. This is a standard practice for tracking algorithms since the algorithms used to estimate the measurement noise (e.g., Newton recursion, LMS, RANSAC, etc.) assume that the noise is Gaussian. However, it is important to note that even though the assumption of Gaussianity is local at each node, the particle distributions are not necessarily Gaussian in the target state space. This will be seen in the simulations. The presented algorithm can handle non-Gaussian noise sources with minor modifications. The assumption of Gaussian noise affects the initialization algorithm at only two stages. The first impact is at the particle proposal stage. Here, if the noise is non-Gaussian, particles can still be generated by importance sampling. The second impact is at the weighting stage. Here, if the pdf of the noise distribution can be evaluated pointwise, then any arbitrary noise distribution can be used. For simplicity, we will assume that the noise is Gaussian in these simulations.

Figures 6(a) to 6(d) represent the sequential particle proposal stage of the initialization algorithm. Although the state vector is four dimensional, the subfigures in Figure 6 show only the *x*-*y* locations of the particles. Nodes 1 and 3 are DOA nodes, whereas nodes 2 and 4 are range-Doppler nodes. There are a few important points to note. First, the total number of particles used to represent the evolving posterior distribution remains constant at each node, regardless of the total number of nodes that provided input to the initialization algorithm. This ensures that the bandwidth remains constant for inter-node communication. It can also be seen that each node that has a detection is equally weighted in the particle representation and the percentage of particles that correspond to a detection from a particular node reduces as the number of nodes that have provided input to the algorithm increases. This is accomplished by the weighted resampling operation explained in Section 2.2.1.3.

Another point to note is that even though each node is incapable of observing the targets' states locally, the evolving particle support is concentrated in parts of the state space where most sensor nodes agree. These final particles are plotted in Figure 6(d) and are propagated back to all the nodes.

Importance weights are calculated for the final particles shown in Figure 6(d). Particles along with their importance weights are used to generate the pdf of the posterior distribution. The *x*-*y* subspace of the posterior distribution is shown in Figure 6(e). As expected, the distribution is highly peaked about the true target states. Estimates of the true target states can be made based on this weighted set of particles. These estimates can be used to initialize any distributed multi-target tracker.

2.3 Low Latency Initialization Algorithm

In this section, we present a low latency implementation of the initialization algorithm given in Section 2.2 that reduces the number of communication passes required for global initialization from three to two: a forward pass to sequentially generate the particle support and importance weights, and a reverse pass to disseminate the final particles and importance weights throughout the network. This algorithm comes at a cost of increased computation at each node but is attractive if the latency incurred as a result of three communication passes is not acceptable. Pseudocode for implementing the low latency algorithm is given in Appendix B.

2.3.1 Theoretical Development

Since the first communication pass is used to sequentially generate the particles and weights, the final set of particles and importance weights available at the output of the m^{th} node, $m = 1, \ldots, M$, must represent the posterior distribution

$$p\left(\mathbf{s}_t | \mathbf{z}_{1,t}, \ldots, \mathbf{z}_{m,t}\right). \tag{28}$$

For the reasons given in Section 2.2.1, (16) is still used as a proposal function and the particle support is generated sequentially at each node. For this choice of proposal function, the importance weights are given by (21). To satisfy these requirements, after processing at node m, particles must be distributed in accordance with

$$\mathbf{s}_t^{(i)} \sim \frac{1}{m} \sum_{\tilde{m}=1}^{m} p\left(\mathbf{s}_t | \mathbf{z}_{\tilde{m},t}\right), \tag{29}$$

and importance weights should be assigned according to

$$w_t^{(i)} \propto \frac{\prod_{\tilde{m}=1}^{m} p\left(\mathbf{s}_t^{(i)} | \mathbf{z}_{\tilde{m},t}\right)}{\sum_{\tilde{m}=1}^{m} p\left(\mathbf{s}_t^{(i)} | \mathbf{z}_{\tilde{m},t}\right)}. \tag{30}$$

Using mathematical induction, we show that it is possible to sequentially generate particles and importance weights according to (29) and (30), respectively, to represent the cumulative posterior distribution (28).

Assume that D particles and importance weights are used to represent the target's state distribution. At node 1, D particles are sampled according to its posterior distribution by following the procedure given in Section 2.2.1. These particles are distributed according to

$$\mathbf{s}_t^{(i)} \sim p\left(\mathbf{s}_t | \mathbf{z}_{1,t}\right). \tag{31}$$

For this choice of particle support, the importance weights are all equal.

$$w_t^{(i)} = \frac{1}{D} \propto \frac{p\left(\mathbf{s}_t^{(i)} | \mathbf{z}_{m,t}\right)}{p\left(\mathbf{s}_t^{(i)} | \mathbf{z}_{m,t}\right)}. \tag{32}$$

It can be clearly seen that (31) and (32) represent (29) and (30), respectively, with $m = 1$. Hence, these particles and importance weights together represent the desired posterior distribution (28) with $m = 1$. This set of particles and importance weights, along with a single number $n_s = 1$ representing the number of nodes that has provided input to the initialization algorithm, is sent to node 2.

Now consider node m. Node m receives a set of particles $\mathbf{s}_{r,t}^{(i)}$, importance weights $w_r^{(i)}$, and n_s from node $(m-1)$. For simplicity of notation, assume that every preceding node

in the chain detects the target and $n_s = (m-1)$. Assume that the received particles are distributed according to

$$\mathbf{s}_{r,t}^{(i)} \sim \frac{1}{m-1} \sum_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_t | \mathbf{z}_{\tilde{m},t}\right), \tag{33}$$

and importance weights are assigned according to

$$w_{r,t}^{(i)} \propto \frac{\prod_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_{r,t}^{(i)} | \mathbf{z}_{\tilde{m},t}\right)}{\sum_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_{r,t}^{(i)} | \mathbf{z}_{\tilde{m},t}\right)}. \tag{34}$$

Thus, the received particles and importance weights together represent the posterior distribution at node $(m-1)$, which is the combined knowledge of the $(m-1)$ nodes that contributed to the initialization algorithm,

$$p\left(\mathbf{s}_t | \mathbf{z}_{1,t}, \ldots, \mathbf{z}_{m-1,t}\right). \tag{35}$$

Node m must now provide its own input to the global posterior distribution. It generates a new set of D particles representing its own posterior distribution

$$\mathbf{s}_{n,t}^{(i)} \sim p\left(\mathbf{s}_t | \mathbf{z}_{m,t}\right). \tag{36}$$

The final set of particles and importance weights after processing at node m must obey (29) and (30). Thus, the new set of particles must be weighted according to

$$w_{n,t}^{(i)} \propto \frac{\prod_{\tilde{m}=1}^{m} p\left(\mathbf{s}_{n,t}^{(i)} | \mathbf{z}_{\tilde{m},t}\right)}{\sum_{\tilde{m}=1}^{m} p\left(\mathbf{s}_{n,t}^{(i)} | \mathbf{z}_{\tilde{m},t}\right)}. \tag{37}$$

Simplifying, we can determine a set of scaled weights $\tilde{w}_{n,t}^{(i)}$ as

$$\tilde{w}_{n,t}^{(i)} = w_{n,t}^{(i)} \cdot \sum_{\tilde{m}=1}^{m} p\left(\mathbf{s}_{n,t}^{(i)} | \mathbf{z}_{\tilde{m},t}\right) \propto p\left(\mathbf{s}_{n,t}^{(i)} | \mathbf{z}_{m,t}\right) \cdot \prod_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_{n,t}^{(i)} | \mathbf{z}_{\tilde{m},t}\right), \tag{38}$$

where each $\tilde{w}_{n,t}^{(i)}$ needs to be adjusted to account for the particle support in order to arrive at the true importance weights given by $w_{n,t}^{(i)}$. In (38), $p\left(\mathbf{s}_t | \mathbf{z}_{m,t}\right)$ is proportional to the data likelihood at node m, given by (24), and $\prod_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_{n,t}^{(i)} | \mathbf{z}_{\tilde{m},t}\right)$ can be approximated using kernel density estimation [24] as follows:

$$\prod_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_{n,t}^{(i)} | \mathbf{z}_{\tilde{m},t}\right) = \sum_{j=1}^{D} w_{r,t}^{(j)} W\left(\mathbf{s}_{n,t}^{(i)} - \mathbf{s}_{r,t}^{(j)}\right). \tag{39}$$

In (39), $W(\cdot)$ is an appropriately chosen stochastic kernel. Different choices of kernel functions will be discussed in Chapter 5.

Using a similar procedure, the particles $\mathbf{s}_{r,t}^{(i)}$ must be weighted according to

$$\hat{w}_{r,t}^{(i)} \propto \frac{\prod_{\tilde{m}=1}^{m} p\left(\mathbf{s}_{r,t}^{(i)}|\mathbf{z}_{\tilde{m},t}\right)}{\sum_{\tilde{m}=1}^{m} p\left(\mathbf{s}_{r,t}^{(i)}|\mathbf{z}_{\tilde{m},t}\right)}. \tag{40}$$

Simplifying, we can determine a set of scaled weights $\tilde{w}_{r,t}^{(i)}$ as

$$\tilde{w}_{r,t}^{(i)} = \hat{w}_{r,t}^{(i)} \cdot \sum_{\tilde{m}=1}^{m} p\left(\mathbf{s}_{r,t}^{(i)}|\mathbf{z}_{\tilde{m},t}\right) \propto p\left(\mathbf{s}_{r,t}^{(i)}|\mathbf{z}_{m,t}\right) \cdot \prod_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_{r,t}^{(i)}|\mathbf{z}_{\tilde{m},t}\right), \tag{41}$$

where $\prod_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_{r,t}^{(i)}|\mathbf{z}_{\tilde{m},t}\right)$ can be approximated using kernel density estimation as follows:

$$\prod_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_{r,t}^{(i)}|\mathbf{z}_{\tilde{m},t}\right) = \sum_{j=1}^{D} w_{r,t}^{(j)} W\left(\mathbf{s}_{r,t}^{(i)} - \mathbf{s}_{r,t}^{(j)}\right). \tag{42}$$

The current set of $2D$ particles $\{\mathbf{s}_{n,t}^{(i)},\ \mathbf{s}_{r,t}^{(i)}\}_{i=1}^{D}$ does not represent (29) since the mixture distribution from which these particles are generated is not equally weighted. This discrepancy can be corrected using a weighted resampling operation similar to the one used in Section 2.2.1. The only modification to be made is that the surviving particles are stored along with their associated scaled weights. The final set of D particles that survive, $\{\mathbf{s}_t^{(i)}\}_{i=1}^{D}$, represent (29).

The scaled weights associated with the particles surviving the resampling operation are stored as $\{\tilde{w}_t^{(i)}\}_{i=1}^{D}$. From (37),(38), (40) and (41), the final set of importance weights representing (30) can be determined by

$$w_t^{(i)} \propto \frac{\tilde{w}_t^{(i)}}{\sum_{\tilde{m}=1}^{m} p\left(\mathbf{s}_t^{(i)}|\mathbf{z}_{\tilde{m},t}\right)}, \tag{43}$$

where $\sum_{\tilde{m}=1}^{m} p\left(\mathbf{s}_t^{(i)}|\mathbf{z}_{\tilde{m},t}\right)$ can be approximated using kernel density estimation as follows

$$\sum_{\tilde{m}=1}^{m} p\left(\mathbf{s}_t^{(i)}|\mathbf{z}_{\tilde{m},t}\right) = \frac{1}{D} \sum_{j=1}^{D} W\left(\mathbf{s}_t^{(i)} - \mathbf{s}_t^{(j)}\right). \tag{44}$$

The final set of particles and importance weights given by $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ obeys (29) and (30). Thus, this weighted set of particles represents the posterior distribution (28).

As stated earlier, the low latency initialization algorithm requires two passes through the communication chain for global initialization: a forward pass to sequentially generate the particle support and importance weights, and a reverse pass to disseminate the final particles and importance weights throughout the network. Data processing occurs only in the first communication pass.

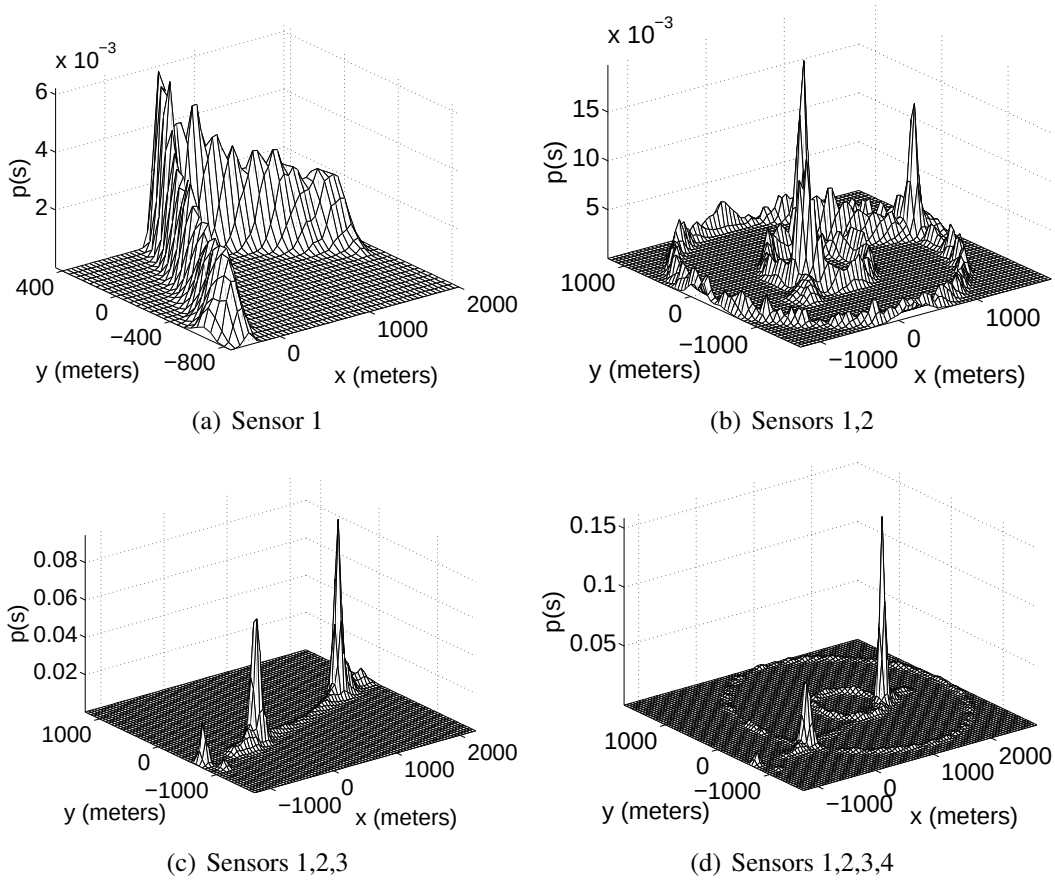
In the first pass, a varying set of a fixed number of D particles representing an equally weighted mixture of the local posteriors of all preceding nodes is transmitted through the communication chain. A set of D importance weights, which along with the set of particles represents the evolving joint posterior of the network, is also transmitted. A single number n_s representing the number of nodes that detected a target and provided their input to the algorithm is transmitted. At the end of the first pass, node M has the final set of D particles that are distributed according to (16), and along with the D importance weights represent the joint network posterior distribution.

In the second pass, the final sets of D particles and D importance weights are propagated throughout the network using the reverse communication path. At the end of the second pass, all nodes share the same particles and importance weights and the network is initialized.

The reduction in inter-node communication passes offered by the low latency algorithm comes at a cost of increased intra-node computation. The bulk of the computational load comes from (39), (42) and (44), which use kernel density estimation. These equations increase the computational complexity to $O(D^2)$ operations, a significant increase from $O(D)$ operations for the low complexity implementation in Section 2.2.1. Thus, the low latency implementation, in its current form, should only be used if the increase in computation is justified by savings in communication.

2.3.2 Simulation

Figure 7 demonstrates the low latency algorithm in operation for the same targets and node locations used in the simulations in Section 2.2.2. The figures show the x-y subspaces of



(a) Sensor 1

(b) Sensors 1,2

(c) Sensors 1,2,3

(d) Sensors 1,2,3,4

Figure 7. Simulation example for initializing multiple targets using the low latency algorithm.

sequentially updated posterior distributions. It can be seen that as more nodes provide input to the algorithm, the posterior distribution slowly has its weight shifted to areas where most nodes are in agreement. The final posterior is shown in Figure 7(d). This distribution is very similar to the posterior obtained in Figure 6(e). The minor differences arise because in the low latency implementation, an approximation of the weighting function is used instead of the true weighting function. However, it is important to note that the dominant peaks remain unchanged regardless of which variant of the algorithm is used.

2.4 Initialization Algorithm with Delay Compensation

Delays can enter the system due to communication, processing and limited signal propagation velocities. In small networks with fast processors at each node, the communication and processing delays can be ignored. However, in heterogeneous networks in which different

sensors operate under different modalities, varying signal propagation velocities can lead to erroneous estimates if they are not compensated. In the simulation examples presented in the earlier sections, the DOA sensors could be acoustic arrays whereas the range-Doppler sensors could use radar. The radar uses electromagnetic signals that propagate at the speed of light and hence its measurements could be assumed to be instantaneous. On the other hand, sound travels at 340m/s. If a target is moving with a high velocity and is at a large distance from the array, then the measurements received at the array could correspond to the target's state at a previous time. In this section, we present an extension to the initialization algorithm in Section 2.2 to compensate for such time delays.

2.4.1 Theoretical Development

Particles $\{\tilde{\mathbf{s}}_t^{(i)}\}_{i=1}^{D}$ are proposed sequentially at each node based on the posterior at that node as given in Section 2.2.1. However, these particles may represent delayed target states and must be shifted in the state space to compensate for the delays before they are propagated to the next node in the communication chain. This movement of particles can be accomplished using a model representing the target's state evolution, as used by the organic tracker at that node, and the estimated delay T. The processing delay, d_{proc}, and the communication delay, d_{comm}, can be estimated at each sensor node beforehand. However, in some cases it may be difficult to estimate the signal propagation delay. For example, in the case of DOA nodes, the range to the target is not observable and although the signal propagation velocity v_m may be known, the true delay cannot be estimated. However, since each particle represents a hypothesized target state, the propagation delay for each particle can be estimated based on the range $r^{(i)}$ to that particle from the current node. Thus, each particle is individually moved to compensate for delays as follows

$$T^{(i)} = \frac{r^{(i)}}{v_m} + d_{\text{proc}} + d_{\text{comm}}, \tag{45}$$

$$\mathbf{s}_t^{(i)} = b_{T^{(i)}}\left(\tilde{\mathbf{s}}_t^{(i)}\right) + T^{(i)}\mathbf{u}_t. \tag{46}$$

The noise added to these final particles accounts for possible target maneuvers, following the state evolution model given in (1). These compensated particles are propagated through the network and the rest of the sequential particle proposal strategy remains unchanged.

Once the final particles are generated, importance weights must be assigned. The weighting function is the same as the one used in Section 2.2.1.4, but if the likelihood function given in (24) is used, then some subset of the particles could be assigned incorrect importance weights since the particles have been moved in accordance with the motion model. The likelihood function is thus modified as follows,

$$p(\mathbf{z}_{m,t}|\mathbf{s}_t^{(i)}) \propto 1 + \frac{1-q}{\sqrt{(2\pi)^s|\tilde{\mathbf{\Sigma}}_{m,T^{(i)}}|}q\lambda K}\times \sum_{k=1}^{K}\exp\left(-\frac{1}{2}(h_{T^{(i)}}(\mathbf{z}_{m,k,t}) - f_m(\mathbf{s}_t^{(i)}))^{\mathrm{H}}\tilde{\mathbf{\Sigma}}^{-1}_{m,T^{(i)}}(h_{T^{(i)}}(\mathbf{z}_{m,k,t}) - f_m(\mathbf{s}_t^{(i)}))\right), \tag{47}$$

$$\tilde{\mathbf{\Sigma}}_{m,T^{(i)}} = \mathbf{J}(h_{T^{(i)}}(\mathbf{z}_{m,k,t}))\mathbf{\Sigma}_{m,t}\mathbf{J}^{\mathrm{H}}(h_{T^{(i)}}(\mathbf{z}_{m,k,t})) + {T^{(i)}}^2\mathbf{\Sigma}_{m,s}, \tag{48}$$

where $\mathbf{\Sigma}_{m,s}$ is the state transition noise covariance matrix for the organic tracker at node m and $h_{T^{(i)}}(\mathbf{z}_{m,k,t})$ is the state transition vector for the organic tracker. For acoustic nodes, using a locally constant velocity model, $h_{T^{(i)}}(\mathbf{z}_{m,k,t})$ is given by

$$h_T(\mathbf{z}_{m,k,t}) = \begin{bmatrix} h_{\theta,T}(\mathbf{z}_{m,k,t}) \\ h_{Q,T}(\mathbf{z}_{m,k,t}) \\ h_{\phi,T}(\mathbf{z}_{m,k,t}) \end{bmatrix}, \tag{49}$$

$$h_{\theta,T}(\mathbf{z}_{m,k,t}) = \tan^{-1}\left(\frac{\sin(\theta_{m,k,t}) + \exp(Q_{m,k,t})T\sin(\phi_{m,k,t})}{\cos(\theta_{m,k,t}) + \exp(Q_{m,k,t})T\cos(\phi_{m,k,t})}\right), \tag{50}$$

$$h_{Q,T}(\mathbf{z}_{m,k,t}) = Q_{m,k,t} - \frac{1}{2}\log\left(1 + 2T\exp(Q_{m,k,t})\cos(\theta_{m,k,t} - \phi_{m,k,t}) + T^2\exp(2Q_{m,k,t})\right), \tag{51}$$

$$h_{\phi,T}(\mathbf{z}_{m,k,t}) = \phi_{m,k,t}. \tag{52}$$

Analytical derivations of (49) through (52) can be found in [33, 30]. For radar nodes, assuming locally constant velocity, $h_{T^{(i)}}(\mathbf{z}_{m,k,t})$ is given by

$$h_T(\mathbf{z}_{m,k,t}) = \begin{bmatrix} h_{r,T}(\mathbf{z}_{m,k,t}) \\ h_{v_r,T}(\mathbf{z}_{m,k,t}) \end{bmatrix}, \tag{53}$$

$$h_{r,T}(\mathbf{z}_{m,k,t}) = r_{m,k,t} + T v_{rm,k,t}, \tag{54}$$

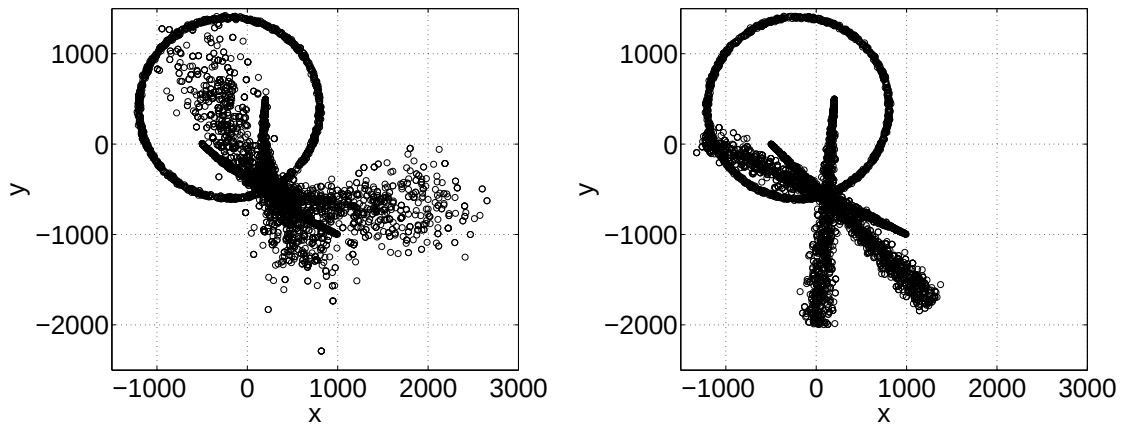
$$h_{v_r,T}(\mathbf{z}_{m,k,t}) = v_{rm,k,t}. \tag{55}$$

Note that the covariance for the data likelihood is given by $\tilde{\mathbf{\Sigma}}_{m,T^{(i)}}$ and consists of two components. The first component depends on the measurement noise. Since future states are predicted using non-linear combinations of elements of $\mathbf{z}_{m,k,t}$ given by $h_T(\mathbf{z}_{m,k,t})$, noise from the components of $\mathbf{z}_{m,k,t}$ is accumulated according to the Jacobian of $h_T(\mathbf{z}_{m,k,t})$. The second component comes from the state transition noise and is introduced to account for the possibility that the targets are maneuvering.

2.4.2 Simulation

To demonstrate the importance and effectiveness of delay compensation, we show simulation results for a fast moving target in a geographically large network. We only simulate the case for a single target so that the effect of compensation is clearly seen in the resulting plots. The target is born at $\mathbf{x}_1 = [250,\ -500,\ 45,\ 35]^{\mathrm{T}}$ and moves at an almost constant velocity. Three acoustic nodes are located at (1000 m, -1000 m), (-500 m, 0 m) and (200 m, 500 m), and a radar node is located at (-200 m, 400 m). The algorithm is simulated for two cases: (a) Compensating for the acoustic propagation delay and (b) Not compensating for acoustic propagation delay. In both cases, we assume that signal propagation is the dominant source of delay in the system, i.e., communication and processing delays are ignored. Figure 8 compares the proposed particles for the two cases.

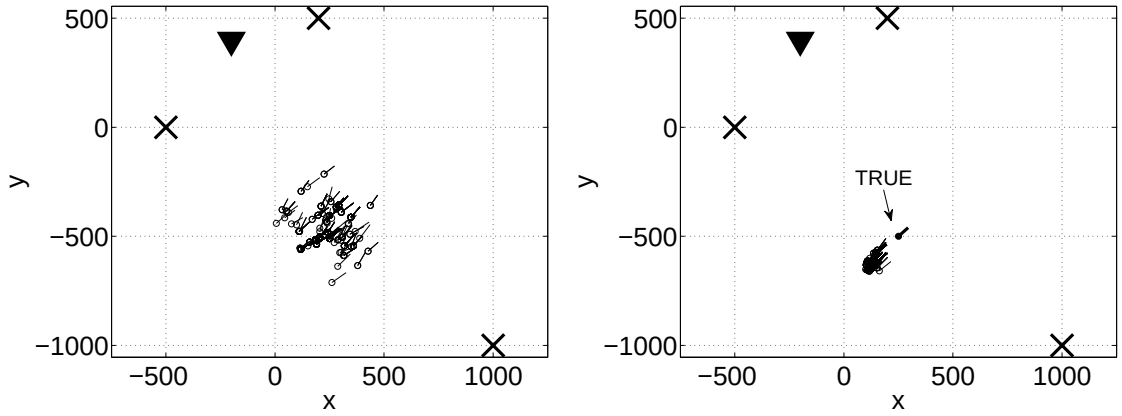
When acoustic signal propagation delay is compensated, particles proposed by the acoustic nodes undergo translation based on the motion model. Particles that are farther away from the node undergo larger translation since the signals from these particles suffer larger propagation delays. This effect is clearly seen in Figure 8(a). In Figure 8(b), however, the proposed particles do not undergo any translation. It can also be seen in Figure 8 that when the signal propagation delay is compensated, the posteriors from the different nodes agree strongly in common areas of the state space when compared to the case when



(a) With delay compensation. (b) Without delay compensation.

Figure 8. Proposed particles with and without delay compensation.

the delay is not compensated. This is because without compensation, the posteriors from the different nodes do not correspond to target states at the same time, leading to biased estimates. This is clearly seen when the final set of weighted particles are sampled based on their importance weights, and the surviving set of particles are plotted for both cases in Figure 9.



(a) With delay compensation. (b) Without delay compensation.

Figure 9. Final set of particles using our initialization algorithm.

Here, the circles represent the particle positions and the lines represent the velocities. These particles can be used to initialize a DJT. The bold marker represents the true target state. For case (a), the resulting particles have mean $\mathbf{x}_C = [261.1,\ -488.6,\ 40.3,\ 40.8]^{\mathrm{T}}$. For case (b), the resulting particles have mean $\mathbf{x}_{NC} = [134.4,\ -625.3,\ 40.9,\ 45.4]^{\mathrm{T}}$. When

compared to the true target state $\mathbf{x}_1 = [250,\ -500,\ 45,\ 35]^{\mathrm{T}}$, it is clear that compensating for the acoustic propagation delay is essential for accurate initialization in large networks with fast targets. It can also be seen in Figure 9 that the final set of particles are spread over a larger area when the delay is compensated, compared to the case when there is no compensation. This is expected since the motion model used to compensate for the propagation delay accounts for possible maneuvers by randomly perturbing the states of the proposed particles. The variance of this random perturbation increases with the length of time for compensation, the variance of the measurement noise, and the variance of the state transition noise. If the target is not expected to maneuver, then the motion model can be altered accordingly and the variance of the final set of particles can be reduced.

2.5 Summary

In this chapter, we have developed various algorithms for distributed sensor network initialization. We started with the low complexity implementation of the initialization algorithm, the theory for which was developed in great detail. This algorithm required three communication passes through the network to achieve global initialization. In some situations, the latency incurred by three communication passes may not be acceptable, and additional computational power may be available at each node to be utilized to reduce the communication load. For such applications, the low latency implementation of the distributed initialization algorithm was developed. This version can achieve global initialization in two communication passes but has increased intra-node computation requirements compared to the low complexity algorithm. A delay compensating implementation of the initialization algorithm was also developed. This version has the ability to compensate for various delays that enter real world systems, including, but not limited to, communication, computation, and limited signal propagation velocities.

Focusing on the distributed data fusion problem, we assumed a simple communication topology. The algorithms developed in this chapter are applicable to sensor networks in

which nodes communicate with each other using a fixed one-hop chain that extends from the first node to the last node in the network. This communication topology is not applicable to most real world sensor networks and will be generalized in the following chapter.

CHAPTER 3

INITIALIZING SENSOR NETWORKS USING COMMUNICATION AND COMPUTATION TREES

3.1 Introduction

In Chapter 2, we focused on the distributed data fusion problem of sensor network initialization, separating it from the specifics of communication topology. Hence, we assumed a very simple communication topology consisting of a fixed one-hop chain extending from the first node to the last node in the network. Limiting the initialization algorithms to such a simple topology has some drawbacks because (i) such a topology is typically not applicable to general sensor networks, and (ii) the fully sequential implementation limits the extent to which processing at the various nodes can be parallelized.

In this chapter, we focus on generalizing the low latency initialization algorithm of Section 2.3 to sensor networks with arbitrary communication topologies. We are not aware of a straightforward method of generalizing the low complexity implementation of Section 2.2 to sensor networks using arbitrary non-chain communication topologies without creating a communication nightmare. On the other hand, generalizing the delay compensating implementation of Section 2.4 requires minimal theoretical development and can be accomplished through rigorous 'book-keeping' of various delays. Hence the details of generalizing the delay compensating algorithm will be overlooked.

We first discuss various communication topologies in general sensor networks. The goal is to find a general topology that is applicable to arbitrary sensor networks. Next, we adapt the distributed initialization algorithm to the general communication topology. We develop our algorithm theoretically and discuss communication and computation demands. Finally, we demonstrate the operation of the algorithm in a surveillance scenario with varying communication topologies using simulated measurements.

3.2 Communication Topologies

Based on connectivity, we can classify sensor networks as connected or disconnected. If it is possible to establish a path through the network that connects any pair of sensor nodes, such a sensor network is said to be connected. If there exists no such path connecting any pair of nodes, then the network is said to be disconnected. In general, disconnected sensor networks can be separated into multiple connected sensor networks that operate independently. Hence, without loss of generality, we can assume that all sensor networks considered in this work are connected. The algorithms developed in this chapter are applicable to connected sensor networks, regardless of the level of connectivity.

Sensor networks can be naturally represented using graphical models. An undirected graph $\mathcal{G}$ consists of a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$ that connect pairs of nodes. If a sensor network is represented by $\mathcal{G}$, then $\mathcal{V}$ represents the set of sensor nodes and $\mathcal{E}$ represents the inter-node communication paths that exist. The edge set will vary depending on the level of connectivity of the network. Two examples of graphs are given in Figure 10.

In Figure 10(a) a connected graph for the 4×4 nearest neighbor grid is shown. It can be seen that multiple cycles exist. If data is shared between neighboring nodes, then it is highly probable that contributions to a message by a particular node can return to the same node after the message travels around the network over various paths. Statistical correlations now exist between the received message and local knowledge, thus breaking the independence assumption needed for the distributed initialization algorithm and affecting the quality of final estimates. This problem has been addressed in the context of the loopy belief propagation (LBP) in [38, 39] where it has been shown that the marginal distributions of the nodes in the network converge to the correct means but the variances are often incorrect. In this dissertation, we attempt to eliminate cycles in the network with the aim of preserving conditional independence.

A *spanning tree* $\mathcal{T}$ of a connected undirected graph $\mathcal{G}$ is a connected graph with no loops. $\mathcal{T}$ is composed of all the nodes $\mathcal{V}$ and a subset of the edges $\mathcal{E}$. Hence all the nodes

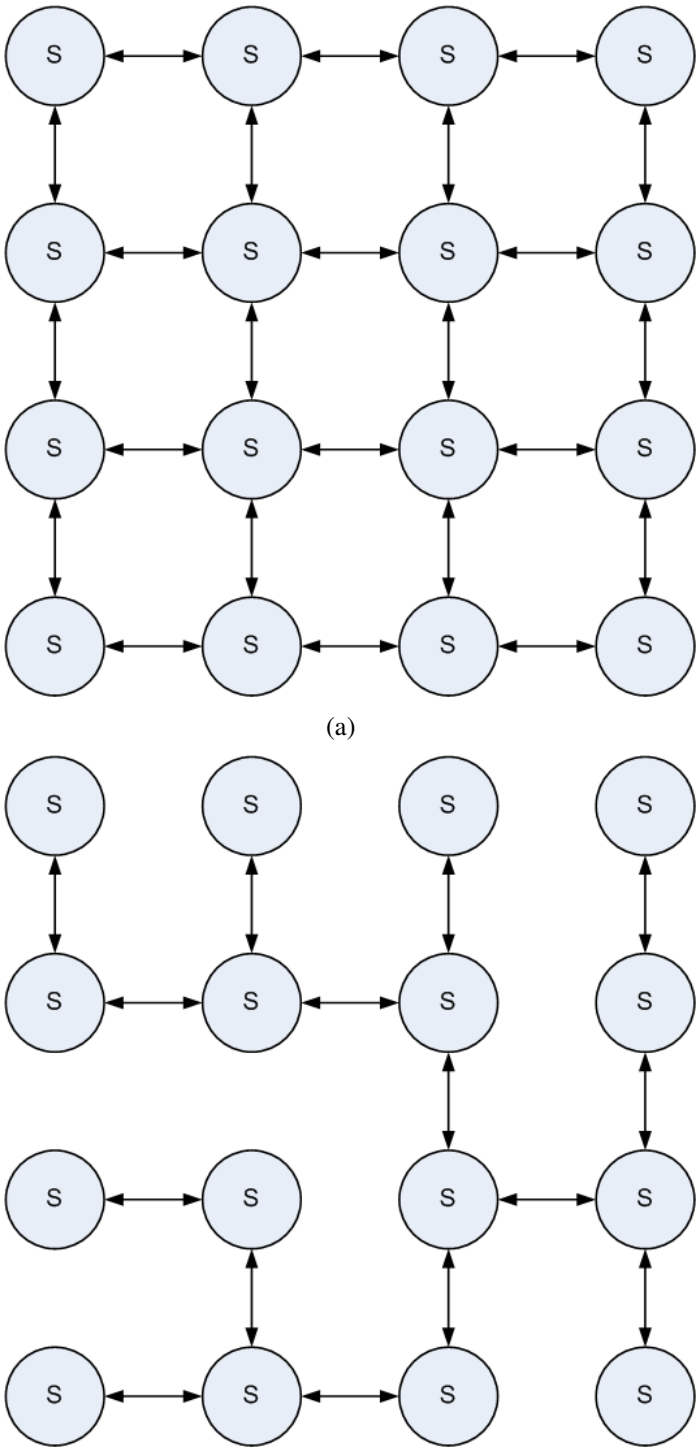


Figure 10. Sample graphical models applied to sensor networks: (a) A 4×4 nearest neighbor grid. (b) A spanning tree for the 4×4 nearest neighbor grid.

are connected without the formation of cycles, by the elimination of a subset of the edges. Any general graph $\mathcal{G}$ might have multiple spanning tree representations. One example of a spanning tree for the 4×4 nearest neighbor grid is shown in Figure 10(b). *Rooting* a tree can be accomplished by assigning any node as the *root* node. The edges of rooted trees may have orientations, typically towards or away from the root. Hence, rooted trees are often treated as *directed acyclic graphs*. Nodes in rooted trees that have no *child* nodes are referred to as *leaf* nodes. In Section 3.3, we develop a distributed initialization algorithm for sensor networks represented by tree structured communication topologies.

The representation of arbitrary connected graphs by spanning trees brings up two interesting questions: (i) how do we determine which node should be assigned as the root node and (ii) which spanning tree representation, if multiple ones exist, should be used. There are various criteria for selecting the root node. One option is to keep the root node fixed at all times. Such a predetermined root node could be selected by virtue of its geographical position, or its sensing and computational power. Another option would be to assign a root node on the fly as the node that is the first to detect an event of interest. The choice of the spanning tree used could be tied to the choice of the root node. Other criteria for choosing the spanning tree include, but are not limited to, minimizing or maximizing the *depth* of the tree, minimizing or maximizing the *breadth* of the tree, and minimizing or maximizing the number of leaf nodes. It is true that different choices of spanning trees and root nodes could affect the amount of processing that occurs in parallel, thus affecting the execution speed of the algorithm. However, the initialization algorithm of Section 3.3 is designed to be unaffected by the order in which nodes provide input to the algorithm, hence making it robust to specific choices of trees and roots.

3.3 Initialization for Tree Structured Networks

The initialization algorithm presented here has general applicability in arbitrary sensor networks because it only depends on the ability of the network to organize its communication

in a tree structure $\mathcal{T}$—any node could play the role of the root node in the tree. A root node would be responsible for launching the initialization algorithm. It would process the data it receives from the environment and produce an estimate of the pdf of the target's state. This estimate is generated in a discrete format using particles and their associated importance weights. These particles and importance weights are then sent to one or more neighboring nodes which are the children of the root node. Each neighboring (child) node generates its own intrinsic set of particles and importance weights based on its organic state estimates. Then a combining step is needed to merge the intrinsic particles and importance weights with the received set of particles and importance weights to produce an improved representation of the target's state distribution. This process is obviously recursive so it can be continued as particles propagate throughout the network from the root node to its descendants (children, grandchildren, and so on). Once the leaf nodes of the tree are reached, it is necessary to propagate the pdf's back up to the root, so that one node will have a pdf that incorporates the measurements from all nodes. In the process of moving the pdf's back up the tree it is necessary to merge pdf's from different branches while not weighting any measurement more than once. The last step in the initialization is a global broadcast of the final pdf from the root to all the other nodes. As stated in [40], such a root-to-leaf-to-root algorithmic structure allows great flexibility. A sample communication tree, with the root node at the top, is given in Figure 11.

3.3.1 Theoretical Development

The fusion algorithms for the downward (root-to-leaf) pass and the upward (leaf-to-root) pass are developed in the following subsections.

3.3.1.1 Downward Pass

Consider a leaf node S_l with parent node P_l and a set of ancestor nodes $\mathcal{GP}_l$. The root node is contained within $\mathcal{GP}_l$. A message launched by the root node propagates downward through the network, collecting data from nodes that it traverses. Note that the message

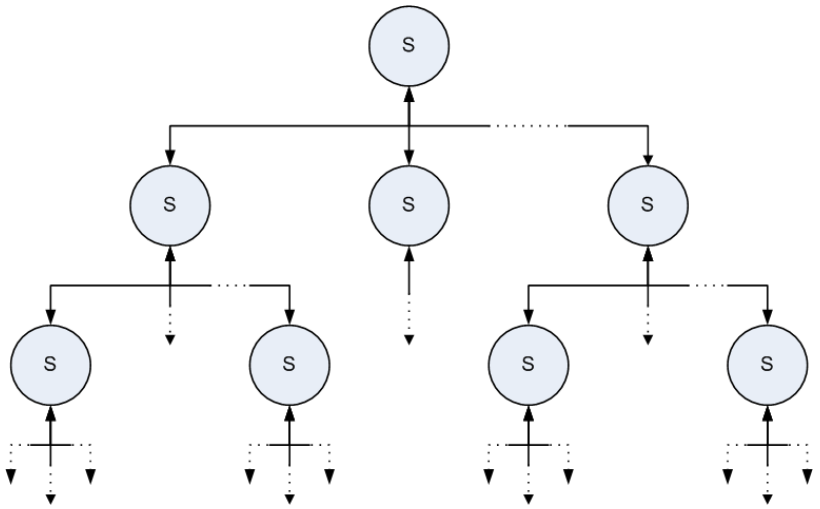


Figure 11. Communication tree. Arrows indicate two-way communication paths.

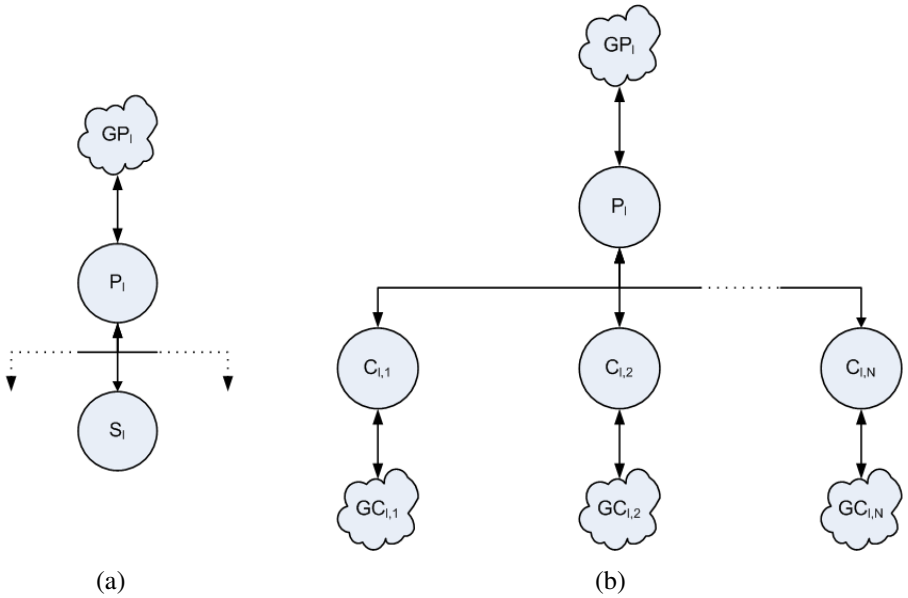


Figure 12. Subtrees for algorithm development. Circles represent single nodes and clouds represent sets of nodes: (a) Downward pass (b) Upward pass.

received at S_l has followed a fixed one-hop communication chain starting from the root node as given in Figure 12(a). This is identical to the communication chain in Figure 4, the distributed fusion algorithm for which has been developed in Section 2.3. After fusing the received message with local organic state estimates, the particles and weights available at S_l are given by (56) and (57), where M_l represents the number of nodes traversed by the

message from the root to S_l.

$$\mathbf{s}_{l,t}^{(i)} \sim \frac{1}{M_l} \sum_{m \in \{S_l, P_l, \mathcal{GP}_l\}} p(\mathbf{s}_t | \mathbf{z}_{m,t}), \tag{56}$$

$$w_{l,t}^{(i)} \propto \frac{\prod_{m \in \{S_l, P_l, \mathcal{GP}_l\}} p(\mathbf{s}_{l,t}^{(i)} | \mathbf{z}_{m,t})}{\sum_{m \in \{S_l, P_l, \mathcal{GP}_l\}} p(\mathbf{s}_{l,t}^{(i)} | \mathbf{z}_{m,t})}. \tag{57}$$

These together represent the distribution

$$p(\mathbf{s}_t | \mathbf{z}_{\mu,t}),\ \mu = \{m | m \in \{S_l, P_l, \mathcal{GP}_l\}\}. \tag{58}$$

3.3.1.2 Upward Pass

While the downward pass of the initialization algorithm was responsible for generating the joint knowledge for individual branches of the tree, the upward pass is responsible for fusing knowledge across branches. The upward pass is more complicated than the downward pass for multiple reasons. First of all, messages received at a parent node from different children may represent the joint knowledge of different numbers of nodes. Secondly, as different branches of the tree may contain some common nodes, the messages received by a parent node from different children may contain some unique knowledge and some redundant knowledge. The common nodes are typically the ancestor nodes contained in the chain extending from the root node to the current parent node. Therefore, care must be taken when fusing the received messages to ensure that all the individual nodes are equally represented.

As shown in Figure 12(b), consider a parent node P_l that receives messages from multiple children nodes $C_{l,n}$, with $n = 1, \ldots, N$. A fusion algorithm developed to effectively fuse messages received by P_l can be used to fuse messages received by any arbitrary node in the network. $\mathcal{GP}_l$ represents the set of ancestor nodes. Without loss of generality, one can assume that a message sent by $C_{l,n}$ to P_l represents the joint knowledge of all subtrees below it, defined by the set of all successor nodes $\{C_{l,n}, \mathcal{GC}_{l,n}\}$, and common knowledge of ancestor nodes $\{P_l, \mathcal{GP}_l\}$. In the following equations, M_* represents the total number

of nodes that have been traversed by the current message and is used simply for normalization. Thus, the message received by P_l from $\mathrm{C}_{l,n}$ consists of particles and importance weights given by

$$\mathbf{s}_{\mathrm{C}_{l,n},t}^{(i)} \sim \frac{1}{M_{l,n}} \sum_{m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,n}, \mathcal{GC}_{l,n}\}} p(\mathbf{s}_t | \mathbf{z}_{m,t}), \tag{59}$$

$$w_{\mathrm{C}_{l,n},t}^{(i)} \propto \frac{\prod_{m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,n}, \mathcal{GC}_{l,n}\}} p(\mathbf{s}_{\mathrm{C}_{l,n},t}^{(i)} | \mathbf{z}_{m,t})}{\sum_{m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,n}, \mathcal{GC}_{l,n}\}} p(\mathbf{s}_{\mathrm{C}_{l,n},t}^{(i)} | \mathbf{z}_{m,t})}. \tag{60}$$

These together represent the distribution

$$p(\mathbf{s}_t | \mathbf{z}_{\mu,t}),\ \mu = \{m | m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,n}, \mathcal{GC}_{l,n}\}\}. \tag{61}$$

Particles and importance weights from the downward pass are still stored at P_l. These are given by

$$\hat{\mathbf{s}}_{\mathrm{P}_l,t}^{(i)} \sim \frac{1}{M_{\mathrm{P}_l}} \sum_{m \in \{\mathcal{GP}_l, \mathrm{P}_l\}} p(\mathbf{s}_t | \mathbf{z}_{m,t}), \tag{62}$$

$$\hat{w}_{\mathrm{P}_l,t}^{(i)} \propto \frac{\prod_{m \in \{\mathcal{GP}_l, \mathrm{P}_l\}} p(\hat{\mathbf{s}}_{\mathrm{P}_l,t}^{(i)} | \mathbf{z}_{m,t})}{\sum_{m \in \{\mathcal{GP}_l, \mathrm{P}_l\}} p(\hat{\mathbf{s}}_{\mathrm{P}_l,t}^{(i)} | \mathbf{z}_{m,t})}. \tag{63}$$

and together represent the distribution

$$p(\mathbf{s}_t | \mathbf{z}_{\mu,t}),\ \mu = \{m | m \in \{\mathcal{GP}_l, \mathrm{P}_l\}\}. \tag{64}$$

After fusion is complete, the final set of particles and importance weights should obey

$$\mathbf{s}_{\mathrm{P}_l,t}^{(i)} \sim \frac{1}{M_l} \sum_{m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,1,\ldots,N}, \mathcal{GC}_{l,1,\ldots,N}\}} p(\mathbf{s}_t | \mathbf{z}_{m,t}), \tag{65}$$

$$w_{\mathrm{P}_l,t}^{(i)} \propto \frac{\prod_{m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,1,\ldots,N}, \mathcal{GC}_{l,1,\ldots,N}\}} p(\mathbf{s}_{\mathrm{P}_l,t}^{(i)} | \mathbf{z}_{m,t})}{\sum_{m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,1,\ldots,N}, \mathcal{GC}_{l,1,\ldots,N}\}} p(\mathbf{s}_{\mathrm{P}_l,t}^{(i)} | \mathbf{z}_{m,t})}. \tag{66}$$

One can define a set of scaled weights as

$$\tilde{w}_t^{(i)} = \prod_{m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,1,\ldots,N}, \mathcal{GC}_{l,1,\ldots,N}\}} p(\mathbf{s}_t^{(i)} | \mathbf{z}_{m,t}). \tag{67}$$

These can be factored as

$$\tilde{w}_t^{(i)} = \prod_{m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,1}, \mathcal{GC}_{l,1}\}} p(\mathbf{s}_t^{(i)} | \mathbf{z}_{m,t}) \prod_{m \in \{\mathrm{C}_{l,2,\ldots,N}, \mathcal{GC}_{l,2,\ldots,N}\}} p(\mathbf{s}_t^{(i)} | \mathbf{z}_{m,t}). \tag{68}$$

Multiplying and dividing by common ancestral knowledge and regrouping terms, we can represent the scaled weights as

$$\tilde{w}_t^{(i)} = \frac{\prod_{n=1}^{N} \left(\prod_{m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,n}, \mathcal{GC}_{l,n}\}} p(\mathbf{s}_t^{(i)} | \mathbf{z}_{m,t}) \right)}{\left(\prod_{m \in \{\mathcal{GP}_l, \mathrm{P}_l\}} p(\mathbf{s}_t^{(i)} | \mathbf{z}_{m,t}) \right)^{N-1}}. \tag{69}$$

Since particles and weights representing the numerator and denominator are given in (59), (60), (62) and (63), the scaled weights can be evaluated using kernel density estimation as

$$\tilde{w}_t^{(i)} = \frac{\prod_{n=1}^{N} \left(\sum_{j=1}^{D} w_{\mathrm{C}_{l,n},t}^{(j)} W\left(\mathbf{s}_t^{(i)} - \mathbf{s}_{\mathrm{C}_{l,n},t}^{(j)}\right) \right)}{\left(\sum_{j=1}^{D} \hat{w}_{\mathrm{P}_l,t}^{(j)} W\left(\mathbf{s}_t^{(i)} - \hat{\mathbf{s}}_{\mathrm{P}_l,t}^{(j)}\right) \right)^{N-1}}. \tag{70}$$

Such scaled weights are evaluated for all received particles and stored as $\{\{\tilde{w}_{\mathrm{C}_{l,n},t}^{(i)}\}_{i=1}^{D}\}_{n=1}^{N}$.

Next, particles representing (65) must be generated from the ND received particles $\{\{\mathbf{s}_{\mathrm{C}_{l,n},t}^{(i)}\}_{i=1}^{D}\}_{n=1}^{N}$. This can be accomplished using a weighted resampling operation. Messages received from each child contain a fixed number of D particles and importance weights, regardless of the number of node posteriors represented by the message. Thus, to ensure that the final particle distribution represents an equally weighted mixture as given in (65), the particles from a message should be assigned resampling weights that are proportional to the number of node posteriors represented by that message. Although it sounds straightforward, the implementation requires some care. Each set of received particles can be divided into two sets: (i) particles representing successor information only that is unique to one of the N received messages and (ii) particles representing common ancestral information that is contained in all received messages. If the particles representing the common ancestor node posteriors are assigned the same resampling weights as the particles representing the

unique successor node posteriors, the final set of particles that survive resampling will over-represent the ancestor posteriors. Hence, the particles representing the common ancestor posteriors must be assigned lower resampling weights.

Let $M_{l,n}$ represents the number of nodes represented in the message received from the n^{th} child $\mathrm{C}_{l,n}$. The first set represents particles sampled from the child and successor posteriors only. This set of particles from $\mathrm{C}_{l,n}$ is given by

$$\{\mathbf{s}^{(j)}_{\mathrm{C}_{l,n},\mathrm{C},t}\} = \{\mathbf{s}^{(i)}_{\mathrm{C}_{l,n},t}\}_{i=1}^{D} \bigcap \overline{\{\hat{\mathbf{s}}^{(i)}_{\mathrm{P}_l,t}\}_{i=1}^{D}}, \tag{71}$$

and assigned resampling weights proportional to the number of node posteriors represented

$$\{\check{w}^{(j)}_{\mathrm{C}_{l,n},\mathrm{C},t}\} = M_{l,n}. \tag{72}$$

The second set represents particles sampled from the parent and ancestor posteriors only. This set of particles from $\mathrm{C}_{l,n}$ is given by

$$\{\mathbf{s}^{(j)}_{\mathrm{C}_{l,n},\mathrm{P},t}\} = \{\mathbf{s}^{(i)}_{\mathrm{C}_{l,n},t}\}_{i=1}^{D} \bigcap \{\hat{\mathbf{s}}^{(i)}_{\mathrm{P}_l,t}\}_{i=1}^{D}. \tag{73}$$

Because each of the N received messages contain particles representing common ancestor node posteriors, their resampling weights are scaled by $1/N$

$$\{\check{w}^{(j)}_{\mathrm{C}_{l,n},\mathrm{P},t}\} = \frac{M_{l,n}}{N}. \tag{74}$$

A set of D particles can be generated by weighted sampling with replacement from the received particles $\{\{\mathbf{s}^{(i)}_{\mathrm{C}_{l,n},t}\}_{i=1}^{D}\}_{n=1}^{N}$ using resampling weights generated using (72) and (74). The surviving particles and the associated scaled weights are labeled and saved as $\{\mathbf{s}^{(i)}_{\mathrm{P}_l,t}, \tilde{w}^{(i)}_{\mathrm{P}_l,t}\}_{i=1}^{D}$.

The scaled weights $\{\tilde{w}^{(i)}_{\mathrm{P}_l,t}\}_{i=1}^{D}$ can now be modified to determine the importance weights $\{w^{(i)}_{\mathrm{P}_l,t}\}_{i=1}^{D}$. From (66) and (67), the importance weights can be defined as

$$w^{(i)}_{\mathrm{P}_l,t} \propto \frac{\tilde{w}^{(i)}_{\mathrm{P}_l,t}}{\sum_{m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,1,\ldots,N}, \mathcal{GC}_{l,1,\ldots,N}\}} p(\mathbf{s}^{(i)}_{\mathrm{P}_l,t} | \mathbf{z}_{m,t})}, \tag{75}$$

and can be evaluated using kernel density estimation as

$$w_{\mathrm{P}_l,t}^{(i)} \propto \frac{\tilde{w}_{\mathrm{P}_l,t}^{(i)}}{\sum_{j=1}^{D} W\left(\mathbf{s}_{\mathrm{P}_l,t}^{(i)} - \mathbf{s}_{\mathrm{P}_l,t}^{(j)}\right)}. \tag{76}$$

The final set of particles and importance weights $\{\mathbf{s}_{\mathrm{P}_l,t}^{(i)}, w_{\mathrm{P}_l,t}^{(i)}\}_{i=1}^{D}$ at P_l represent the desired joint distribution

$$p(\mathbf{s}_t|\mathbf{z}_{\mu,t}),\ \mu = \{m|m \in \{\mathcal{GP}_l, \mathrm{P}_l, \mathrm{C}_{l,1,\ldots,N}, \mathcal{GC}_{l,1,\ldots,N}\}\}, \tag{77}$$

and are propagated to the parent node for P_l. This fusion procedure is repeated at each node during the upward pass until messages arrive at the root node. After fusing incoming messages at the root node, the final particles and importance weights $\{\mathbf{s}_t^{(i)}, w_t^{(i)}\}_{i=1}^{D}$ represent the joint distribution for the entire network.

3.3.1.3 Communication and Computation

The generalized initialization algorithm presented in this chapter requires three passes through the selected spanning tree for global initialization: a root-to-leaf pass, or downward pass, to sequentially generate the particle support and importance weights representing the joint distribution for individual branches, a leaf-to-root pass, or upward pass, to fuse distributions across branches, and a final downward pass to disseminate the final particles and importance weights throughout the network. Data processing occurs only in the first two communication passes.

The downward pass is identical to the first pass of the low latency initialization algorithm of Section 2.3. Hence, the message passed from a parent node to each of its N children consists of D particles, D importance weights, and a single number n_s representing the number of nodes that detected a target and provided their input to the algorithm. The computational load at each node is $O(D^2)$.

In the upward pass, the message transmitted from each child node to its parent node consists of D particles, D importance weights, and a single number n_s representing the number

of nodes that detected a target and provided their input to the algorithm. Determining the scaled weights in (70) requires $O(N^2D^2)$ operations and dominates the computational load at each node in the upward pass.

The final downward pass is needed to broadcast the final sets of D particles and D importance weights throughout the network. No computation takes place during this final downward pass.

3.3.2 Simulations

The generalized initialization algorithm is simulated for a network consisting of bearing nodes and range-Doppler nodes as shown in Figure 13. For the same reasons given in Chapter 2, none of these nodes are capable of locally observing the state vectors of two targets, T1 and T2, that appear simultaneously in the network. A total of 7 nodes exist, S1 through S7. All the odd numbered nodes are bearing nodes and all the even numbered nodes are range-Doppler nodes. The algorithm developed in Section 3.3.1 is used to fuse noisy local estimates in a distributed manner, with minor modifications made to account for missed detections. Explicit data association is not required. In the following simulations, we demonstrate the performance of the initialization algorithm with varying communication paths and target occlusions.

3.3.2.1 Simulation I

In the first simulation, the inter-node communication paths are as given in Figure 14. S1 is the root node that launches the initialization algorithm. The other nodes perform their functions sequentially. Organizing this network hierarchically, the paths taken by various messages form the tree given in Figure 15. In the downward pass, nodes communicate in the order in which they have been numbered. In the upward pass, nodes communicate in the reverse order. For this simulation, we assume that all 7 nodes detect both targets.

Figure 16 shows the evolving particle support in *x-y* space as messages propagate through the network during the downward pass from the root node towards the leaf nodes.

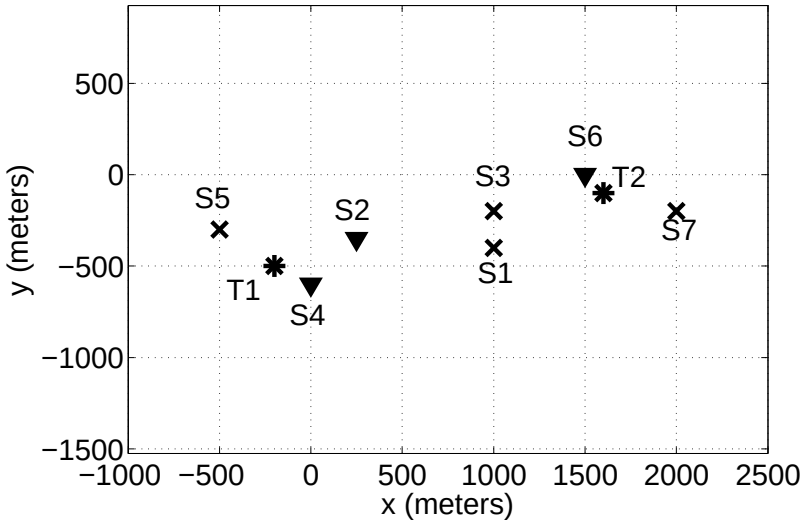


Figure 13. Sensor network setup. $\times$ represent bearing nodes, $\triangledown$ represent range nodes, and $*$ represent targets.

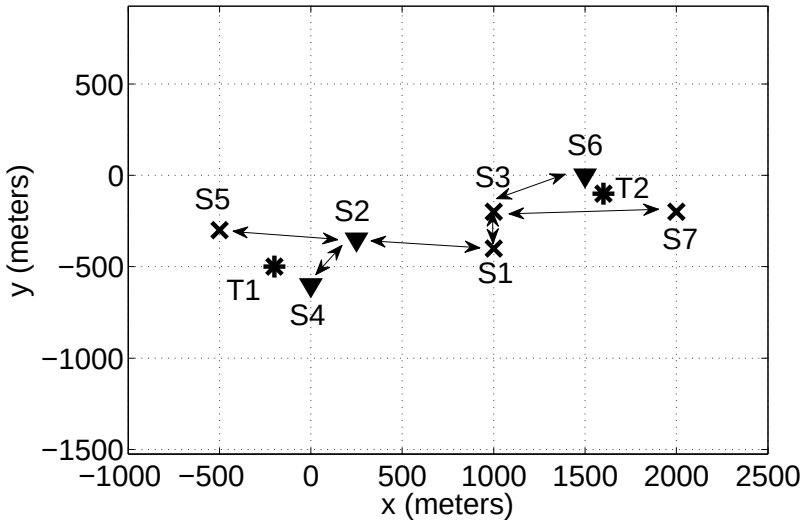


Figure 14. Network with tree communication. $\times$ represent bearing nodes, $\triangledown$ represent range nodes, $*$ represent targets, and $\leftrightarrow$ represent communication paths.

Note that even though the particles contain velocity information and each particle has an associated importance weight, this information is not displayed. As messages propagate downwards, information is fused over the branches of the tree being traversed. As seen in the particle support at each of the leaf nodes, the fused data represents the combined

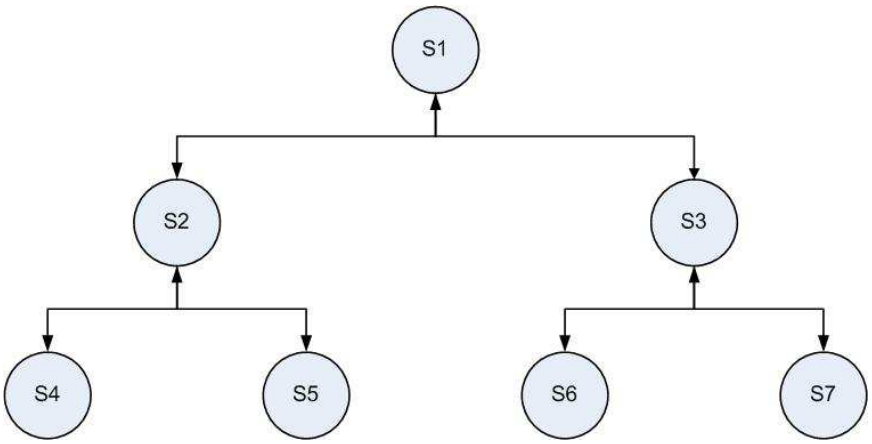


Figure 15. Communication tree for the simulated network

knowledge of the chain of sensor nodes from the root to that particular leaf. For example, the particle distribution at S7 represents the combined knowledge of the chain S1-S3-S7 consisting of three bearing nodes. Data still needs to be fused across the various branches to represent the joint knowledge over the entire network. This is accomplished in the upward pass.

Figure 17 shows the evolving particle support in *x-y* space as messages propagate through the network during the upward pass from the leaf nodes towards the root node. As messages propagate upwards, knowledge from multiple child nodes are merged by the parents and redundant ancestral knowledge is compensated so it is not over-represented. The final set of particles available at the root node represent an equally weighted mixture of posteriors from all the nodes in the network.

The joint distribution for the entire network generated using the particles and weights at the end of the upward pass is plotted in Figure 18. It can be seen that two distinct peaks appear at the true target locations. This demonstrates that the initialization algorithm developed in this paper is effective in fusing data across the network.

3.3.2.2 Simulation II

Since the initialization algorithm developed in this chapter is intended to be a generalization of the basic initialization algorithm developed for communication chains in Chapter 2,

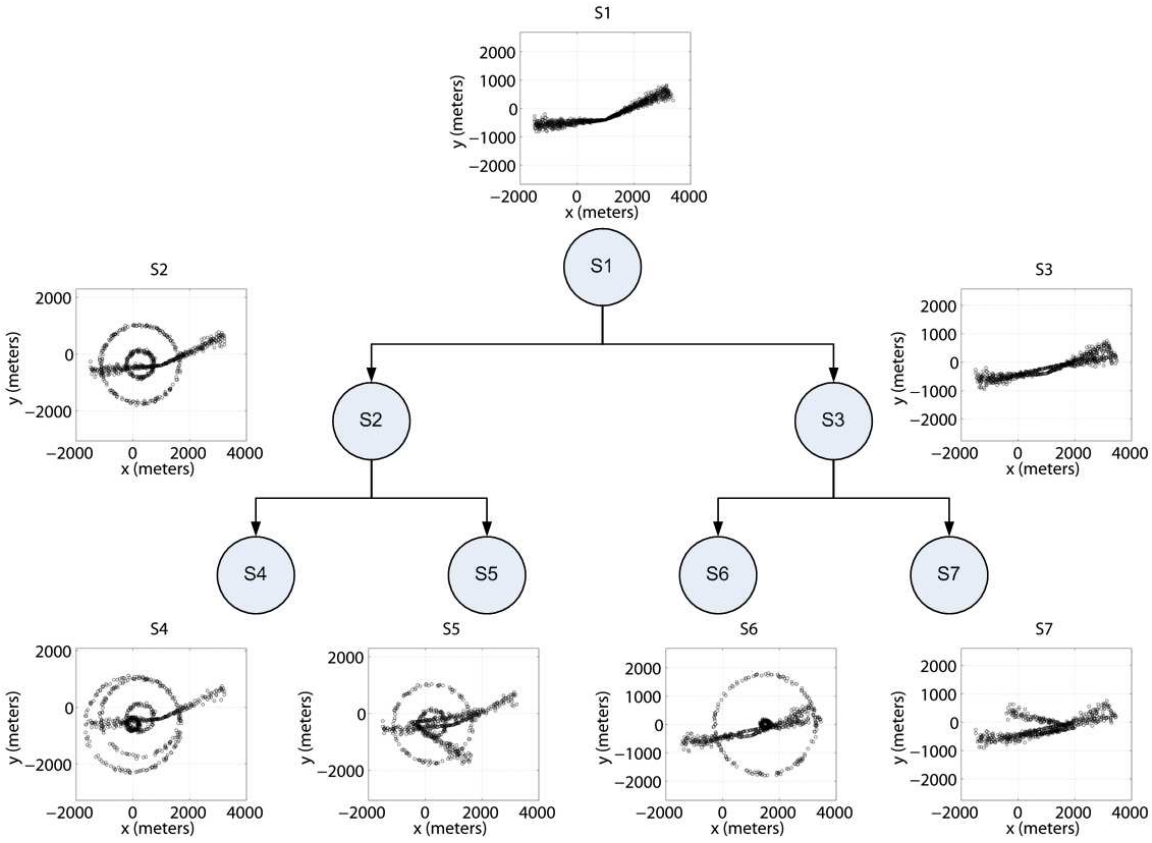


Figure 16. Particle distribution during the downward pass through the communication tree when all sensors detect both targets.

simulations are repeated for the same network in Figure 13 with a chain communication topology as given in Figure 19. The evolution of the particle distribution is given in Figure 20. Because the network is organized as a chain, all the data in the network is fused by the end of the downward pass. The upward pass is simply used to propagate the final particles and weights back to the root node. The final set of particles and weights at S7 is used to generate the joint network posterior distribution given in Figure 21.

Comparing these results to those of Simulation I, two important features stand out. First, it can be seen that the final set of particles at S7 in Figure 20 closely resembles the final set of particles at S1 in Figure 17. This is expected since the final particle distribution is an equally weighted mixture of local posterior distributions from all the nodes in the network and is statistically unaffected by the order in which nodes provide input to the algorithm. The minor differences are due to specific random number realizations used to

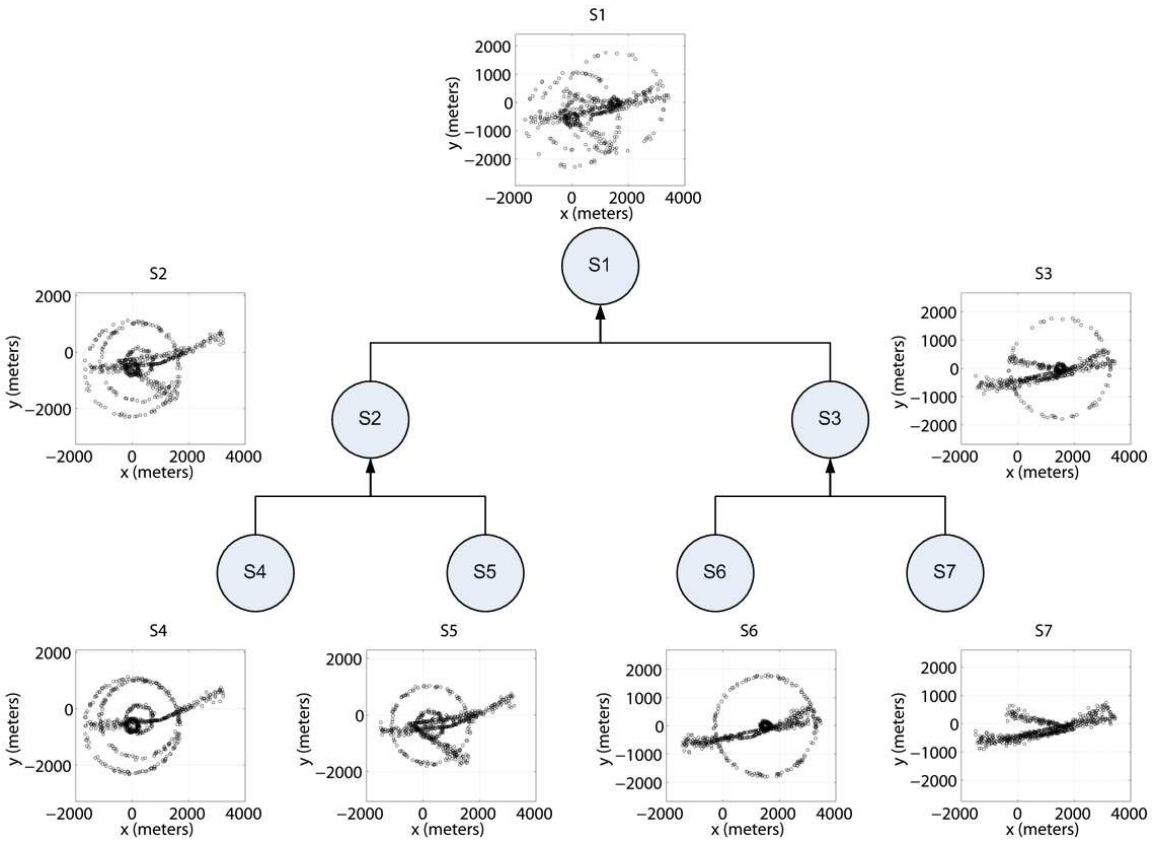


Figure 17. Particle distribution during the upward pass through the communication tree.

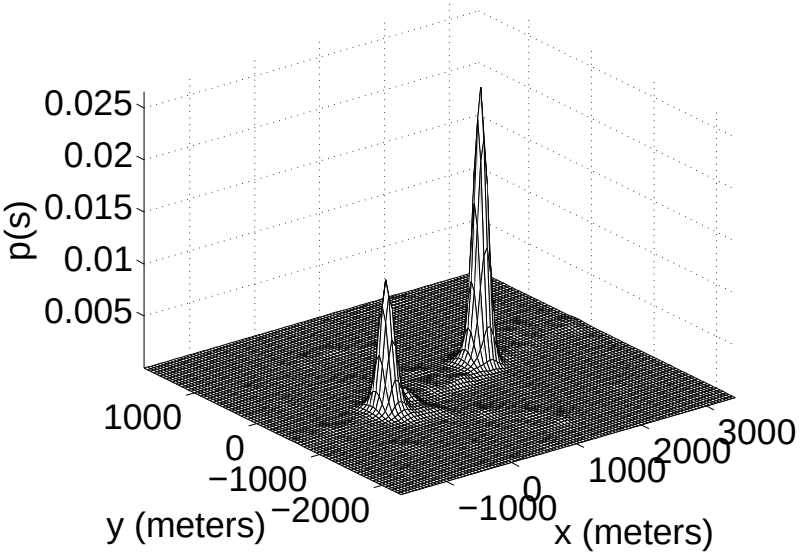


Figure 18. Joint distribution for the network using a communication tree. All sensors see both targets.

generate the particle support at each node. Another feature is that the posterior distributions in Figures 18 and 21 are both multimodal and have peaks at the true target locations. This is

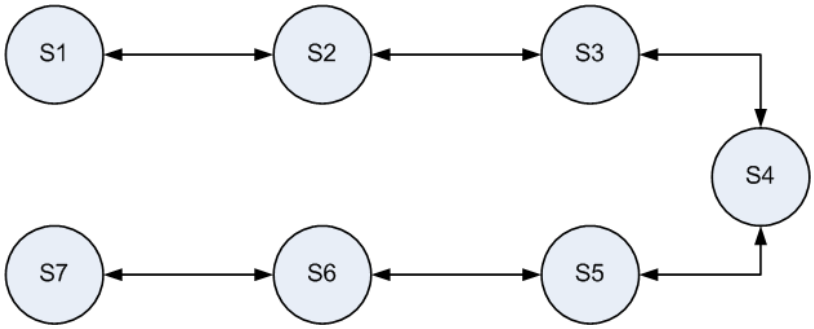


Figure 19. Communication chain for the simulated network.

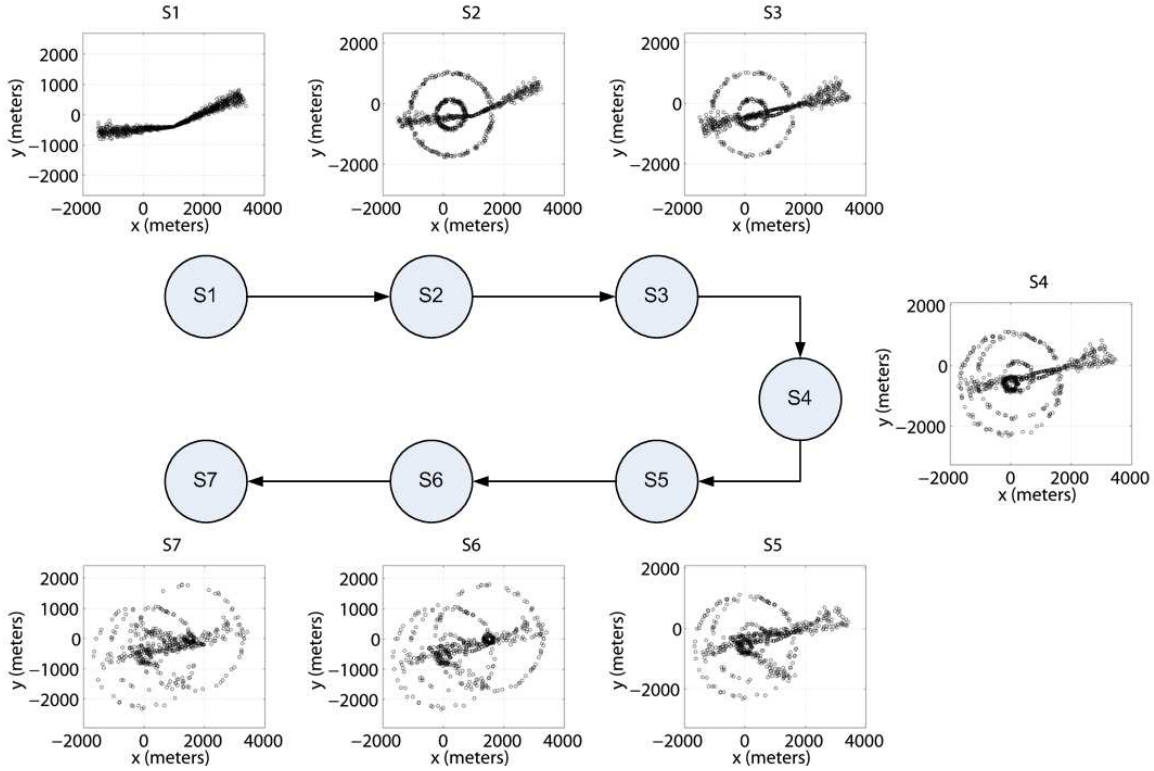


Figure 20. Particle evolution during the downward pass through a communication chain.

also expected as the final joint network posterior distribution is proportional to the product of local posterior distributions and is statistically unaffected by node scheduling. The minor differences in the low probability regions are due to specific realizations of the random number generators and are dropped from our consideration.

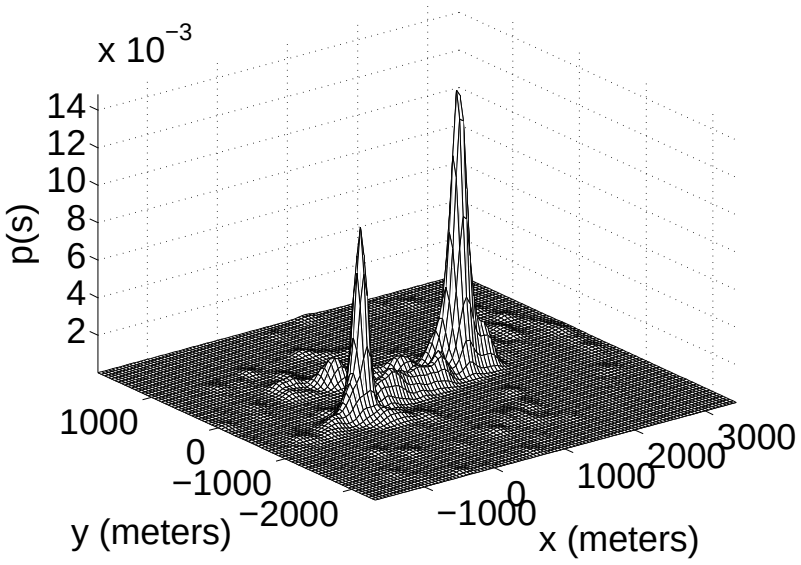


Figure 21. Joint distribution for the network using a communication chain.

3.3.2.3 Simulation III

In this simulation, we run our initialization algorithm on yet another communication topology in the same sensor network given in Figure 13. A communication tree of unit height is created with S1 as the root node and all other nodes as leaf nodes connected directly to S1. We refer to this communication topology as a *spider* topology. The graphical representation of the spider topology is given in Figure 22.

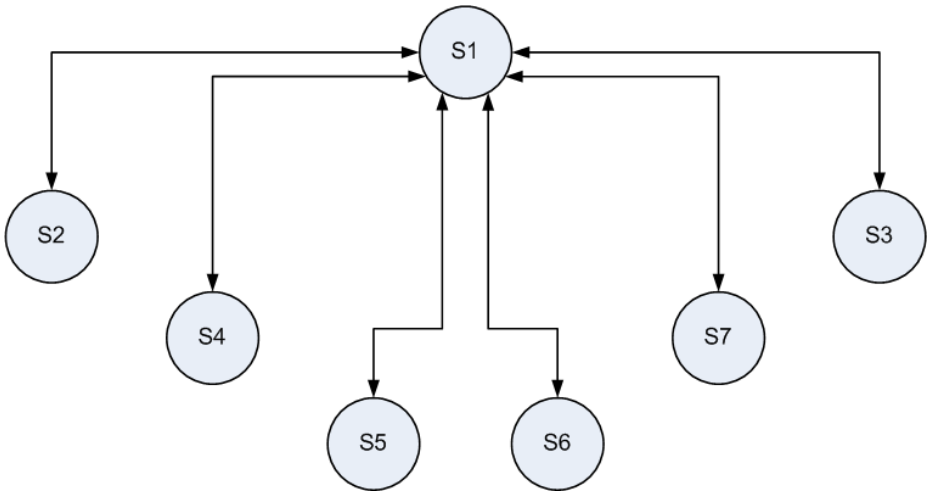


Figure 22. Spider topology represents a communication tree with unit height.

The evolution of the particle distribution during the downward and upward passes are

given in Figures 23 and 24 respectively. The final set of particles and weights available at

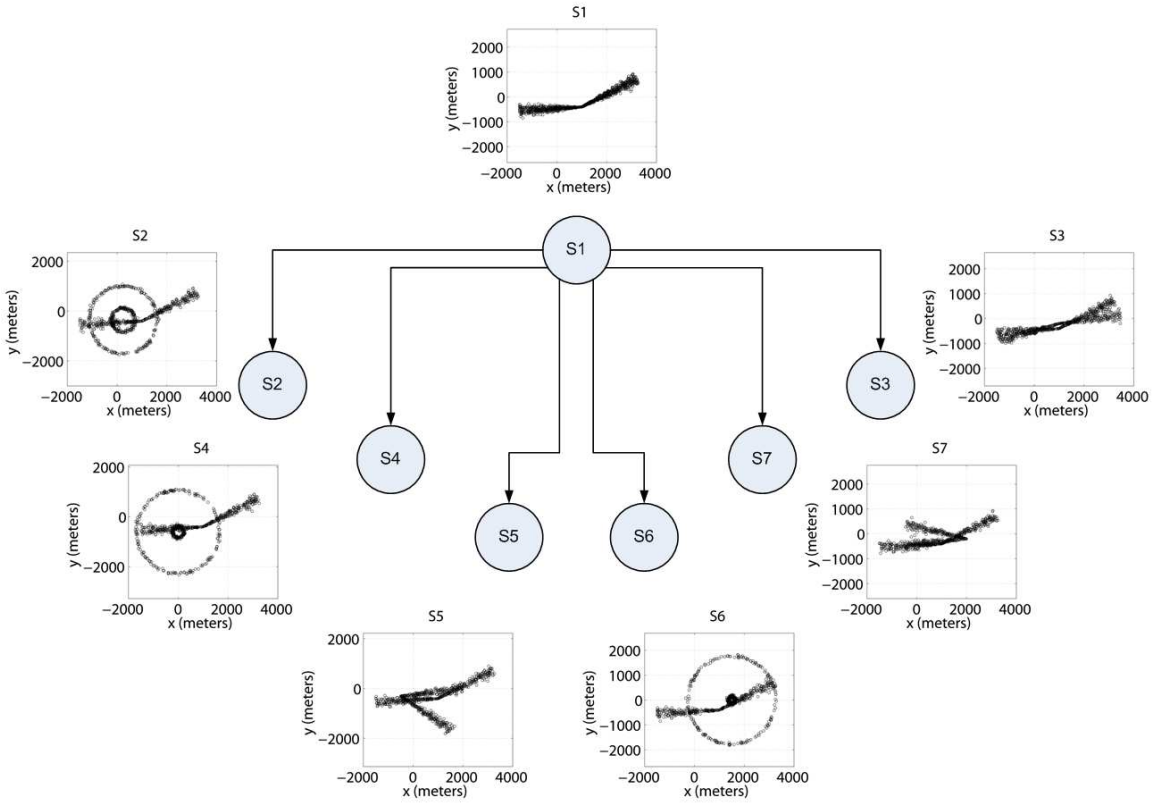


Figure 23. Particle evolution during the downward pass through a spider topology.

S1 at the end of the upward pass is used to generate the joint network posterior distribution given in Figure 25. Comparing these results to those of simulations I and II, we see consistent performance regardless of the particular topology used by the network for inter-node communication. Consistency is seen in the final particle distribution, available at S1 in Figures 17 and 24, and at S7 in Figure 20, as well as in the final posterior distributions given in Figures 18, 21 and 25. Once again, the minor differences are due to specific realizations of the random number generators. This further goes to show that our initialization algorithm is unaffected by specific communication topologies.

3.3.2.4 Simulation IV

In this simulation, we use the same communication topologies as given in simulations I, II and III, but we shift our focus to how the initialization algorithm performs with target

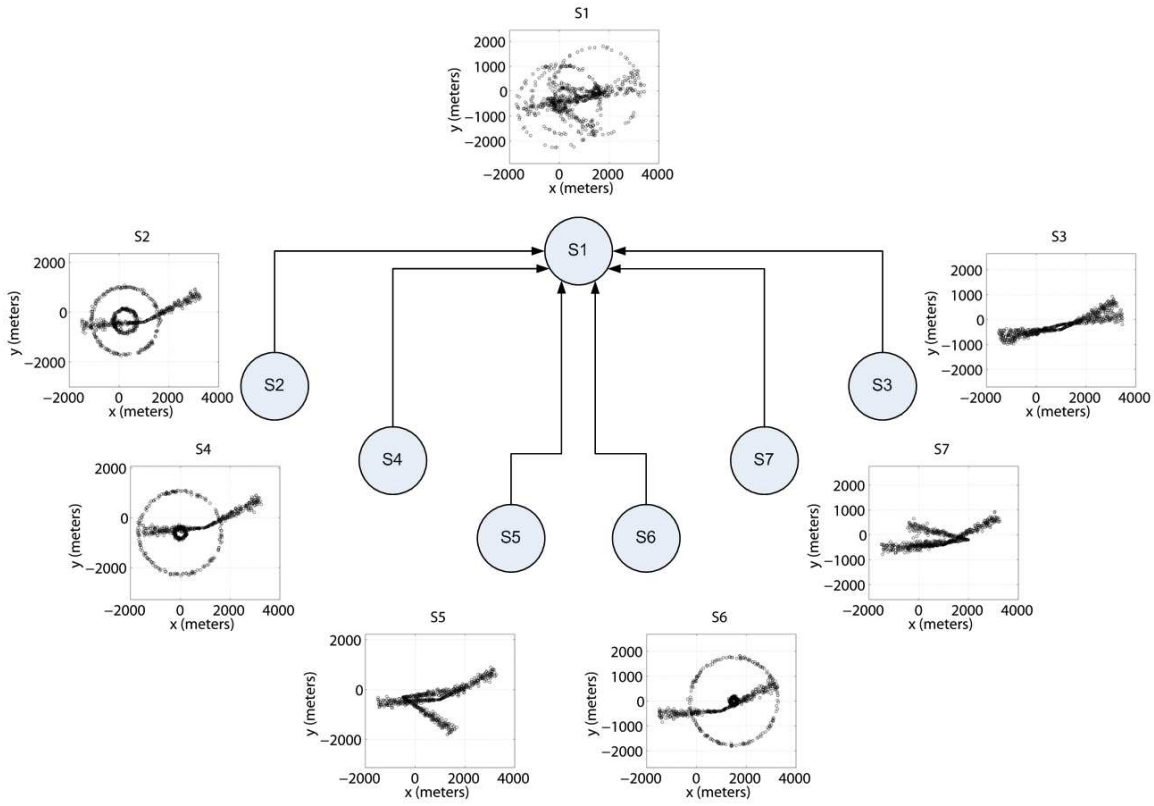


Figure 24. Particle evolution during the upward pass through a spider topology.

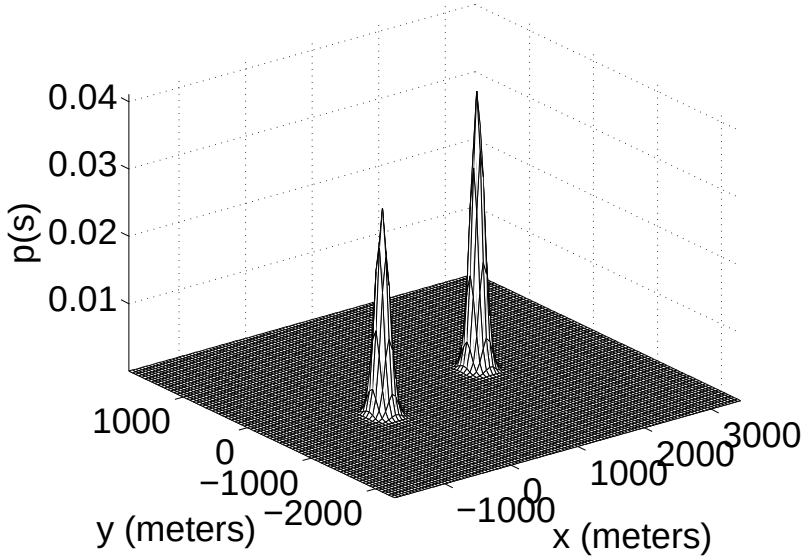


Figure 25. Joint distribution for the network using a spider topology.

occlusions. Therefore, in the simulated measurements available at each sensor, not all sensors are capable of detecting both targets. Only S1 is capable of detecting both targets. S2, S4 and S5 only detect T1, whereas S3, S6 and S7 only detect T2. Once again, in the downward pass, nodes communicate in the order in which they have been numbered, whereas in the upward pass, nodes communicate in the reverse order.

We first simulate the network using a communication tree as given in Figure 15. The evolving particle distributions during the downward and upward passes are shown in Figures 26 and 27, respectively. Due to target occlusions, the local posterior distributions at

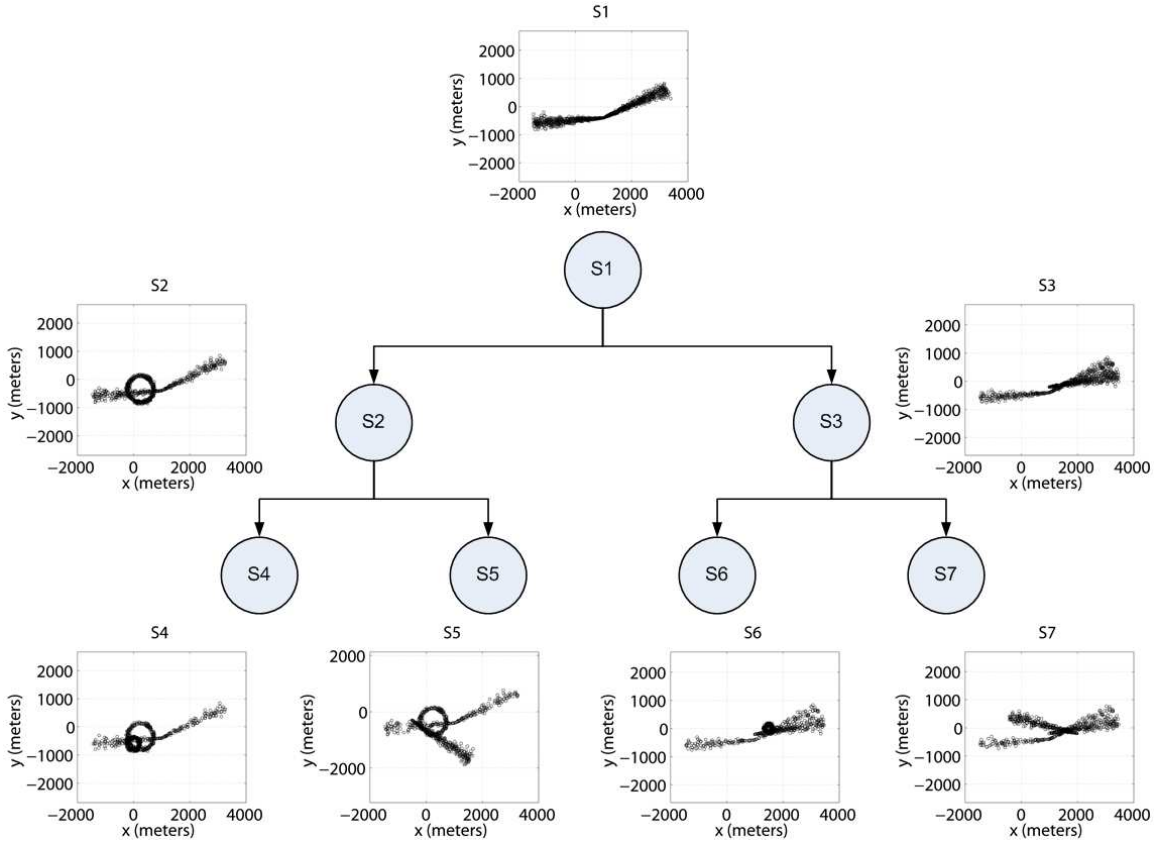


Figure 26. Particle distribution during the downward pass through the communication tree with target occlusions.

the individual nodes in the network differ from the local posterior distributions in simulation I. This results in a different final particle distribution available at S1 at the end of the upward pass. These particles and their associated importance weights are used to generate the pdf of the joint network posterior distribution given in Figure 28. Despite targets being

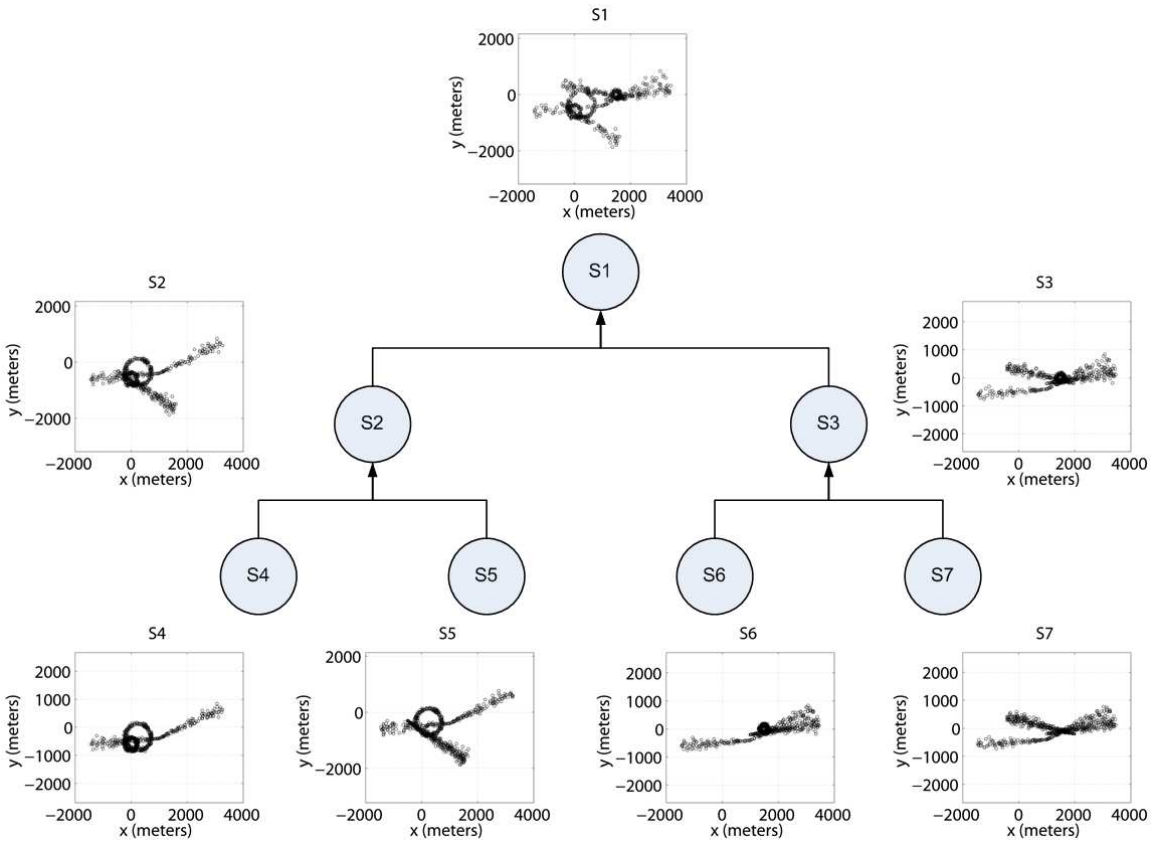


Figure 27. Particle distribution during the upward pass through the communication tree with target occlusions.

occluded at a subset of the sensors in the network, the final joint distribution is multimodal with distinct peaks at the true target locations. Hence, in this simulation, it can be seen that our initialization algorithm is robust to missed detections at a subset of the sensors in the network. In general, the posterior distribution will be peaked at locations in the target state space where most sensors are in agreement, regardless of which sensor sees which target.

We simulate the same target occlusions discussed above in networks using the chain and spider communication topologies as given in Figures 19 and 22 respectively. We do not show the sequential particle generation, but the final network posterior distributions in each case are given in Figures 29 and 30. It can be seen that both posterior distributions are multimodal with peaks at the true target locations. This demonstrates consistent performance of our initialization algorithm regardless of the particular communication topology used.

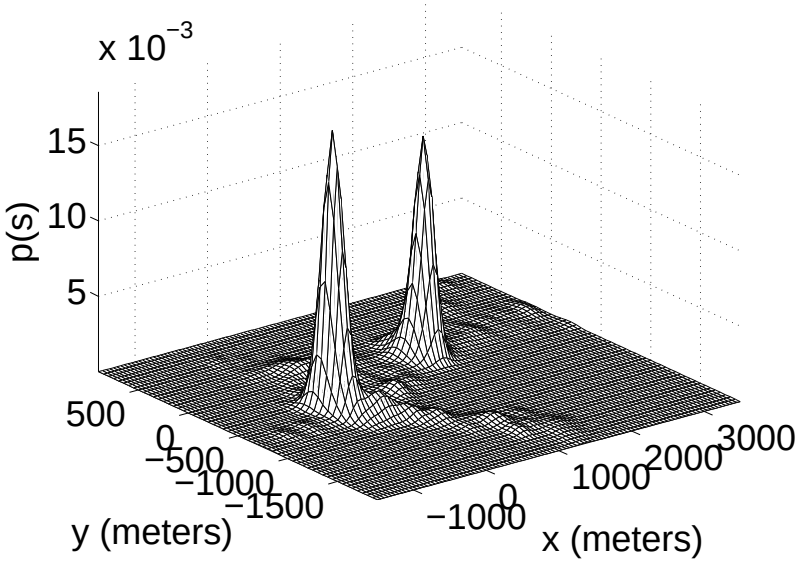


Figure 28. Joint distribution for the network using a communication tree with occluded targets.

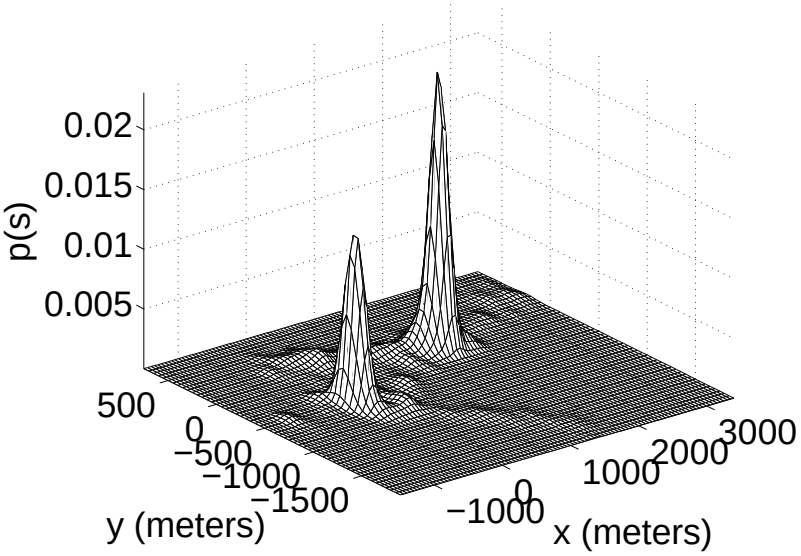


Figure 29. Joint distribution for the network using a communication chain with occluded targets.

3.4 Summary

In this chapter, we have generalized the initialization algorithm of Chapter 2 to sensor networks with arbitrary communication topologies. We showed that any sensor network that is globally connected can be represented using a spanning tree. We then developed the theory for the generalized initialization algorithm for sensor networks using arbitrary communication trees. This algorithm was implemented in a simulated sensor network with

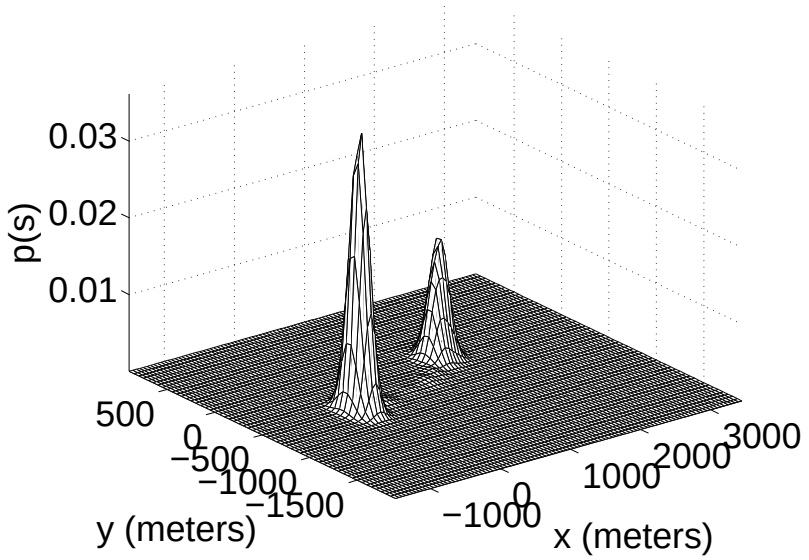


Figure 30. Joint distribution for the network using a spider communication topology with occluded targets.

varying communication topologies and target occlusions.

In the computational analysis of the generalized initialization algorithm, which is given in Section 3.3.1.3, we have shown that the majority of the computational load appears in the upward communication pass. With D particles and D weights representing messages transmitted from each of the N children to their parent node, the upward pass requires $O(N^2D^2)$ operations at each parent node. It is clear that the particular choice of spanning tree representation for the sensor network could significantly affect the computational load at each node, which increases quadratically with the number of children. However, having multiple children allows parallel processing, thereby offering the opportunity to reduce any processing latency that may exist in the system. The selection of the optimal spanning tree is out of the scope of this dissertation but is an exciting topic for future research.

CHAPTER 4

COMPARISON WITH BELIEF PROPAGATION METHODS

4.1 Introduction

Distributed estimation of the evolving hidden states of a system has traditionally been addressed in a Bayesian framework by using Belief Propagation (BP) methods [41]. In BP methods, the estimated target posterior distribution can be communicated through the network and it evolves as the various sensor nodes provide their input to the estimate. Using BP methods in the tracking problem, electrical engineers, computer scientists, and statisticians handle the distributed communication issues by only passing messages to neighboring nodes and by casting the tracking problem in a Markov random field (MRF) framework [42] as shown in Figure 31.

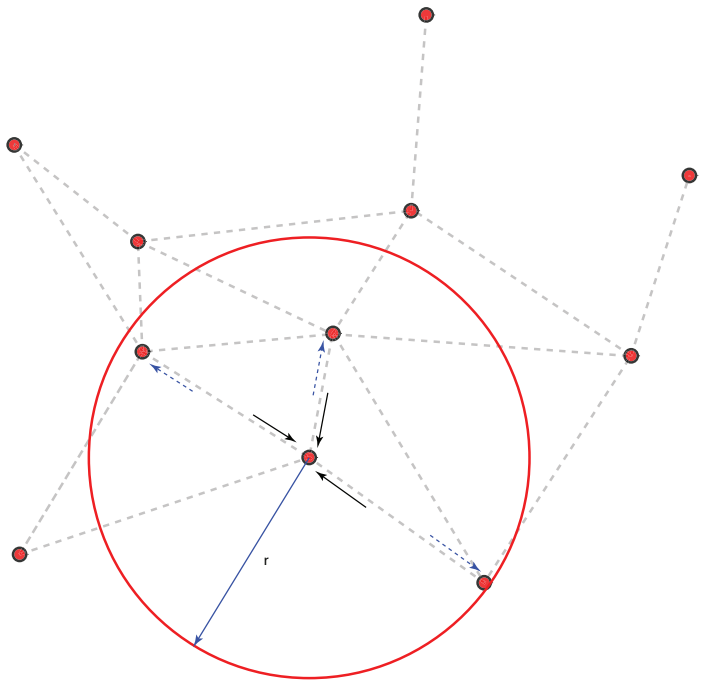


Figure 31. In the BP framework, when the information at a sensor node is transmitted to neighboring nodes as messages, the communication can be achieved using a constant channel bandwidth. To preserve the node's battery life, the messages are passed only to those neighbors within its transmission range r_t. Because of the data processing that occurs in the node to represent the cumulative state information using constant bandwidth, information can only be lost [43]. Hence, we need to trade-off the communication bandwidth for more precise state representations.

For the reasons given in Section 1.2, the analytical evaluation of complicated integrals in the BP equations may not be feasible. Therefore, two almost identical nonparametric methods were independently developed as solutions to this problem. One method is called nonparametric belief propagation (NBP) [44] and the other is called particle message passing (PAMPAS) [27]. Although these methods are quite similar, they differ in how messages are created, and this leads to differences in performance.

We begin this chapter with a brief overview of the theory and equations involved in BP and motivate the need for nonparametric implementations. Next, we use BP to address the problem of sensor network initialization and compare it to our distributed initialization algorithm. Both, theoretical analysis and simulation results are provided

4.2 Overview of Belief Propagation

Detailed discussions of NBP and PAMPAS are given in [44, 27] respectively. For the reader's convenience, we provide the basic formulation in this section. Consider a set of nodes, $\mathcal{V}$, and a set of edges, $\mathcal{E} = \{(n_1, n_2) \mid n_1, n_2 \in \mathcal{V}\}$, that define an undirected graph, $\mathcal{G}$. If a sensor network is represented by $\mathcal{G}$, then $\mathcal{V}$ represents the set of sensor nodes in the network and $\mathcal{E}$ represents the communication paths between various nodes. We can define the neighborhood, $\mathcal{P}(s)$, of a node, $s \in \mathcal{V}$, as the set of nodes, $t \in \mathcal{V}$, that satisfies $(s,t) \in \mathcal{E}$. Each node, s, is associated with a hidden state variable, $\mathbf{s}_s$, and an observed measurement, $\mathbf{y}_s$. We can define *node potentials*, $\psi_s(\mathbf{s}_s, \mathbf{y}_s)$, that represent the relationship between the state variable and the observed measurement at a node, and *edge potentials*, $\psi_{st}(\mathbf{s}_s, \mathbf{s}_t)$, that represent the relationship between the state variables of neighboring nodes. The joint distribution of the state variables and measurements is given by

$$p(\mathbf{s}, \mathbf{y}) = \frac{1}{Z} \prod_{s \in \mathcal{V}} \psi_s(\mathbf{s}_s, \mathbf{y}_s) \prod_{(s,t) \in \mathcal{E}} \psi_{st}(\mathbf{s}_s, \mathbf{s}_t), \tag{78}$$

where $\mathbf{s}$ represents the network state vector, and Z is used for normalization so that the integral of (78) equals unity. The goal of BP methods is to find the conditional marginal distributions, $p(\mathbf{s}_s|\mathbf{y})$, for all the nodes in the network. This is accomplished by sharing

messages between neighboring nodes. The message sent from node s to node t is given by

$$m_{st}(\mathbf{s}_t) = \alpha \int_{\mathbf{s}_s} \psi_{st}(\mathbf{s}_s, \mathbf{s}_t)\psi_s(\mathbf{s}_s, \mathbf{y}_s) \prod_{u \in \mathcal{P}(s)\setminus t} m_{us}(\mathbf{s}_s) d\mathbf{s}_s, \tag{79}$$

where α is a normalizing constant. An approximation to the marginal distribution at a node is obtained by combining the incoming messages with the local measurement as

$$p(\mathbf{s}_t|\mathbf{y}) = \alpha\psi_t(\mathbf{s}_t, \mathbf{y}_t) \prod_{u \in \mathcal{P}(t)} m_{ut}(\mathbf{s}_t). \tag{80}$$

By passing messages between nodes and with an appropriate choice of the state vector, BP methods can be used to initialize distributed sensor networks.

Since the analytical evaluation of complicated BP equations may not be feasible for non-Gaussian potentials, NBP and PAMPAS were developed as nonparametric alternatives. In these methods, the messages propagated between nodes consist of D particles, and possibly their associated importance weights, and continuous approximations to the messages are made using KDE. Thus, each message is approximated by a mixture density that is generated by placing appropriate kernels at each particle location. Typically, diagonal covariance Gaussian kernels are used. Messages received from neighboring nodes are combined by multiplying them together. The product of the received messages is then multiplied by local knowledge.

Assume that we have N Gaussian densities that are multiplied together. For the n^{th} density, let μ_n represent the mean, Σ_n represent the covariance, and w_n represent the relative weighting. The product density is also a Gaussian with covariance $\bar{\Sigma}$, mean $\bar{\mu}$, and weight $\bar{w}$ given by

$$\bar{\Sigma}^{-1} = \sum_{n=1}^{N} \Sigma_n^{-1}, \tag{81}$$

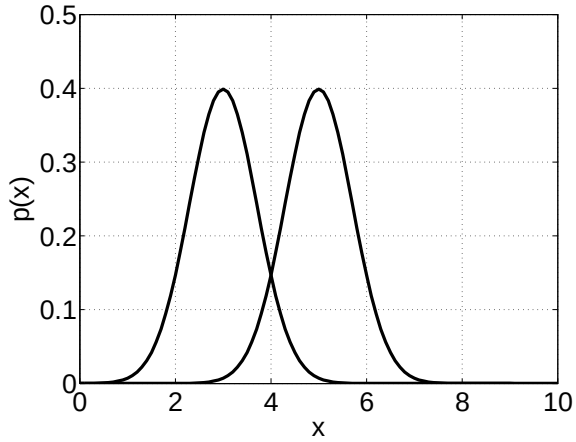
$$\bar{\mu} = \bar{\Sigma} \sum_{n=1}^{N} \Sigma_n^{-1}\mu_n, \tag{82}$$

$$\bar{w} \propto \frac{\prod_{n=1}^{N} w_n \mathcal{N}(\mu_n, \Sigma_n)}{\mathcal{N}\left(\bar{\mu}, \bar{\Sigma}\right)}. \tag{83}$$

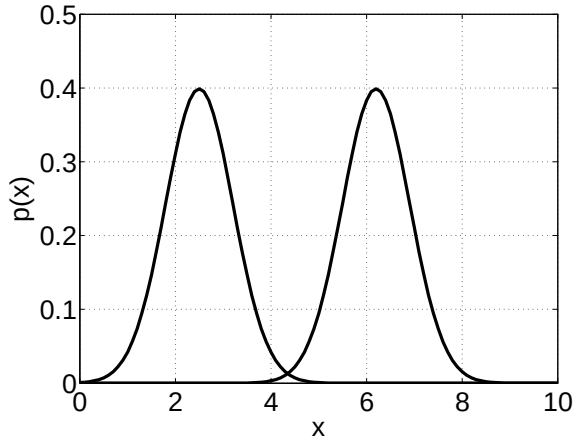
For the general case of N input mixtures consisting of D components each, direct multiplication results in a mixture consisting of D^N components. A resampling step is required to maintain a fixed bandwidth for inter-node communication and also to maintain computational tractability. Various algorithms have been proposed to select a representative sample from the D^N product components and prevent the total number of mixture components from growing exponentially. A naive exhaustive method would first explicitly compute all D^N product densities. Then, a weighted sampling with replacement operation would be performed according to the relative weights $\bar{w}$ of the product components to generate an equally weighted mixture of D components. To illustrate the relative weight assignment, consider a simple example of two one-dimensional mixture densities, each consisting of two equally weighted Gaussian components. These mixture densities, along with their four component product mixture, are shown in Figure 32. It can be seen that the product components have higher weighting in regions of the state space where the input densities have overlapping components. The greater the amount of overlap, the higher is the assigned weighting. These define the high probability regions of the state space. After a weighted sampling with replacement operation according to the assigned weights, the product components with high weights are likely to survive, whereas those with low weights are likely to be eliminated. Thus, the surviving mixture components are likely to be concentrated in the high probability regions of the state space.

In [44], the Gibbs sampler is used to select a representative sample from the product mixture without explicitly computing all D^N product components. Use of the Gibbs sampler reduces the computational load to $O(kND^2)$ operations, where k represents the number of iterations required by the Gibbs sampler to generate one sample. In [45], mixture importance sampling is proposed as a method to reduce computation to $O(ND^2)$ operations. [45] also proposes the use of kd-trees, the computational cost of which is dependent on the choice of various approximating parameters.

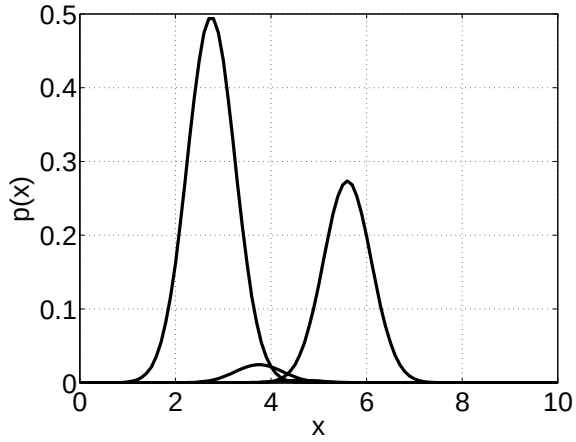
Although NBP and PAMPAS both use particle sets to represent messages, there are



(a) Input mixture 1.



(b) Input mixture 2.



(c) Product mixture.

Figure 32. Taking products of Gaussian mixture densities. The input mixture densities in (a) and (b) contain 2 components each. The product mixture in (c) contains 4 components, one of which is assigned such a small weight that it is barely visible.

subtle differences that lead to differences in performance. Whereas PAMPAS is specialized for graphical models in which the potentials can be expressed as a mixture of Gaussians, NBP allows more general potential functions. However, the variance estimates generated by NBP tend to be biased above their true values, while those generated by PAMPAS are unbiased.

4.3 Using Belief Propagation for Distributed Initialization

4.3.1 Theoretical Formulation

The nonparametric implementations of BP find direct applicability to our distributed initialization problem. The node potentials are now simply the local posterior distributions,

$$\psi_s(\mathbf{s}, \mathbf{y}_s) = p(\mathbf{s}|\mathbf{y}_s). \tag{84}$$

Since we are interested in estimating the joint network posterior distribution and not the local marginal distributions, the state vector to be estimated has the same parameters at each node. Therefore, the edge potentials are meaningless and can be dropped. Equations (79) and (80) can now be rewritten as

$$m_{st}(\mathbf{s}) = \alpha p(\mathbf{s}|\mathbf{y}_s) \prod_{u \in \mathcal{P}(s) \setminus t} m_{us}(\mathbf{s}), \tag{85}$$

$$p(\mathbf{s}|\mathbf{y}) = \alpha p(\mathbf{s}|\mathbf{y}_s) \prod_{u \in \mathcal{P}(s)} m_{us}(\mathbf{s}), \tag{86}$$

Since the primary difference between NBP and PAMPAS lies in how new messages are generated using the edge potentials, this difference is eliminated in our application and facilitates a common implementation, which we will refer to as the *BP implementation*.

Comparing the BP implementation to our initialization algorithm, various similarities can be seen. Both algorithms attempt to produce a discrete approximation to the joint network posterior distribution. The approximation is obtained by sequentially taking the products of local node posterior distributions and propagating these products through the network. Both algorithms maintain fixed bandwidth for inter-node communication. The

primary difference lies in how the joint distribution is represented. In our initialization algorithm, the particles are distributed according to an equally weighted mixture of local node posterior distributions, and importance weights are assigned to represent the evolving joint network posterior. In the BP implementation, particles are themselves distributed according to the evolving joint network posterior and are concentrated in areas of high probability. Thus, in a simple sense, the importance function we use in our initialization algorithm has an additive form whereas the importance function used by the BP implementation has a multiplicative form. Both methods have advantages and disadvantages.

A major advantage of the BP implementation is that it can potentially use fewer particles than our initialization algorithm. This is because as the joint network posterior distribution evolves, the BP implementation constantly shifts particles to high probability regions of the state space. Our initialization algorithm, on the other hand, retains local posterior distributions in the particle support. Hence, a large number of particles could lie in low probability regions of the joint network posterior distribution. This is particularly true if sensors in the network lack local observability of the target state. The problem worsens with an increase in the number of sensors in the network.

However, the multiplicative form of the BP implementation can lead to some major disadvantages. Since the estimate of the posterior distribution is generated by multiplying messages together, errors in the final estimate can arise when one or more of the messages are erroneous. For example, if a node is confident in an erroneous estimate, then it could incorrectly pull the particle distribution towards an incorrect state. Our algorithm is expected to show greater robustness to false alarms since the particle distribution retains local knowledge. A similar effect is seen in the case of missed detections. For example, if the evolving posterior distribution has two peaks corresponding to two targets, and the current node detects only one of the two targets, then all the particles used by the BP implementation could be shifted to the region of the state space corresponding to that detected target.

Depending on the communication topology and node scheduling, even if all the other sensors in the network detect both targets, the particle distribution may never be pulled back to the second target's state. We show that our initialization algorithm demonstrates greater robustness to missed detections, communication topology and node scheduling.

4.3.2 Simulations

In this section we simulate the BP implementation for comparison against our distributed initialization algorithm. To ensure a fair comparison, the BP implementation was simulated under circumstances that were as close as possible to those under which our initialization algorithm was simulated. We simulate the same 7 sensor network with the same 2 targets used in Section 3.3.2. For convenience, the network setup is once again shown in Figure 33. A total of 1000 particles are used in each simulation. We implemented a sequential

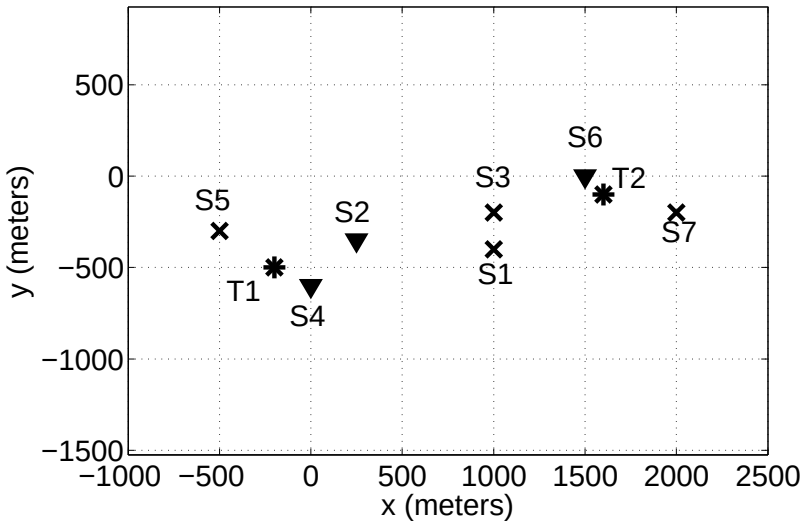


Figure 33. Sensor network setup. $\times$ represent bearing nodes, $\triangledown$ represent range nodes, and $*$ represent targets.

schedule for inter-node communication as discussed in [46]. This schedule is very similar to the one-hop communication protocol used to simulate our initialization algorithm, and is applicable to the chain and tree communication topologies.

For acyclic communication topologies, estimates of the joint network posterior distribution generated using BP converge to the true posterior distribution once the messages from each node reach every other node in the network [44]. Using the chain and tree topologies, this condition is satisfied at the end of the forward pass and at the end of the upward pass, respectively. In practice, multiple iterations may be required for the nonparametric implementation of BP to converge. However, we only simulate a single iteration to make a fair comparison with our algorithm. Following [27], a single outlier particle with zero mean and large covariance was included with each message. This outlier particle was weighted to account for 10% of the entire message probability, which is identical to the probability of miss in the simulation of our initialization algorithm. Among the various methods available to reduce computations, we use mixture importance sampling [45] to determine message products at each node.

4.3.2.1 Simulation I

In this simulation, we compare our distributed initialization algorithm with the BP implementation when each sensor detects both targets. We use the chain communication topology given in Figure 34. The sequential particle proposal stages for both algorithms are

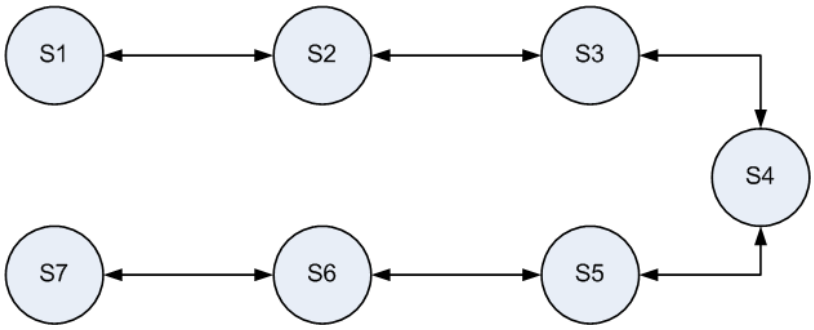


Figure 34. Communication chain for the simulated network.

shown in Figures 35 and 36. In Figure 35, it is clear that the multiplicative form of the importance function used by the BP implementation shifts the particle support to areas of high probability. Therefore, at the end of the forward pass, the particles at S7 are concentrated around the true target states. On the other hand, the final particle distribution generated by

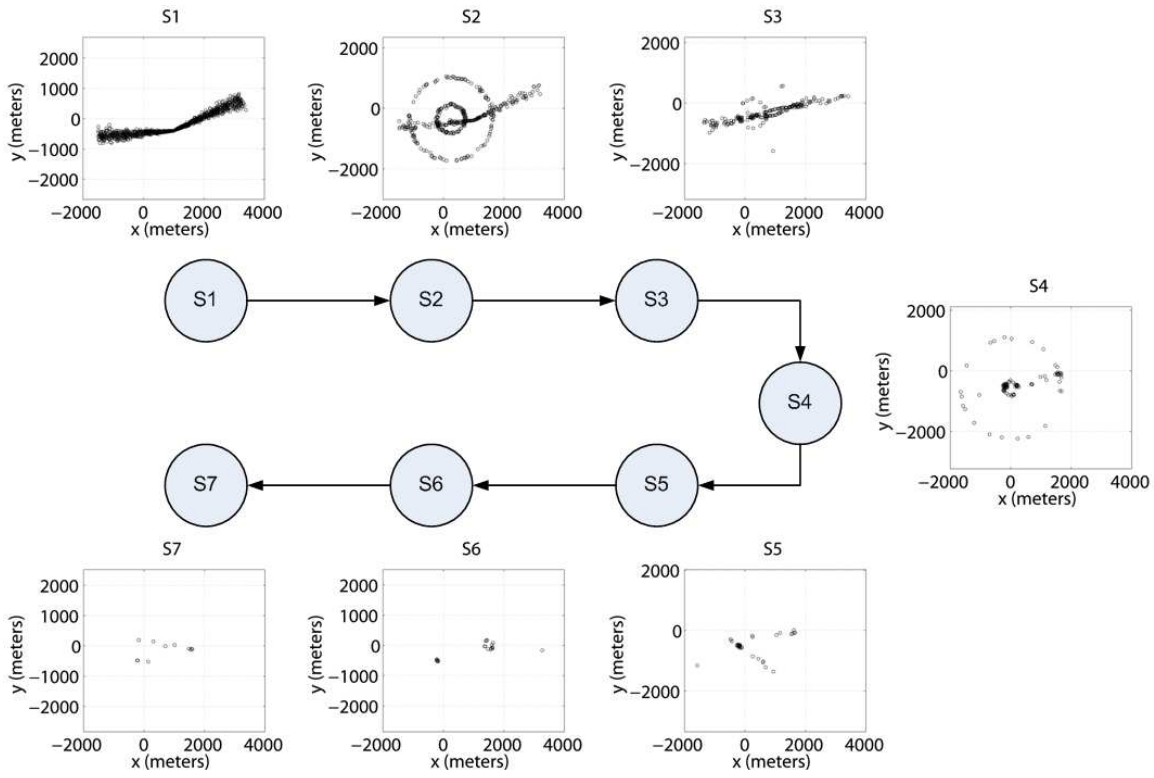


Figure 35. Particle evolution during the downward pass through a communication chain using the BP implementation.

our distributed initialization algorithm has a large number of particles in low probability regions of the state space. This behavior can be seen in the particle distribution at S7 in Figure 36.

The final posterior distributions generated using our initialization algorithm and the BP implementation are shown in Figure 37. Both distributions have strong peaks at the true target states, demonstrating satisfactory performance from both algorithms when used to initialize new targets. However, the particle distribution generated by our initialization algorithm results in small bumps in the low probability regions of the joint network posterior compared to that generated by the BP implementation. Such behavior is commonly seen in densities generated by KDE if data points lie in the tails of the approximated densities [24]. Since we are only interested in finding the peaks of the joint network posterior distribution, this behavior can be neglected.

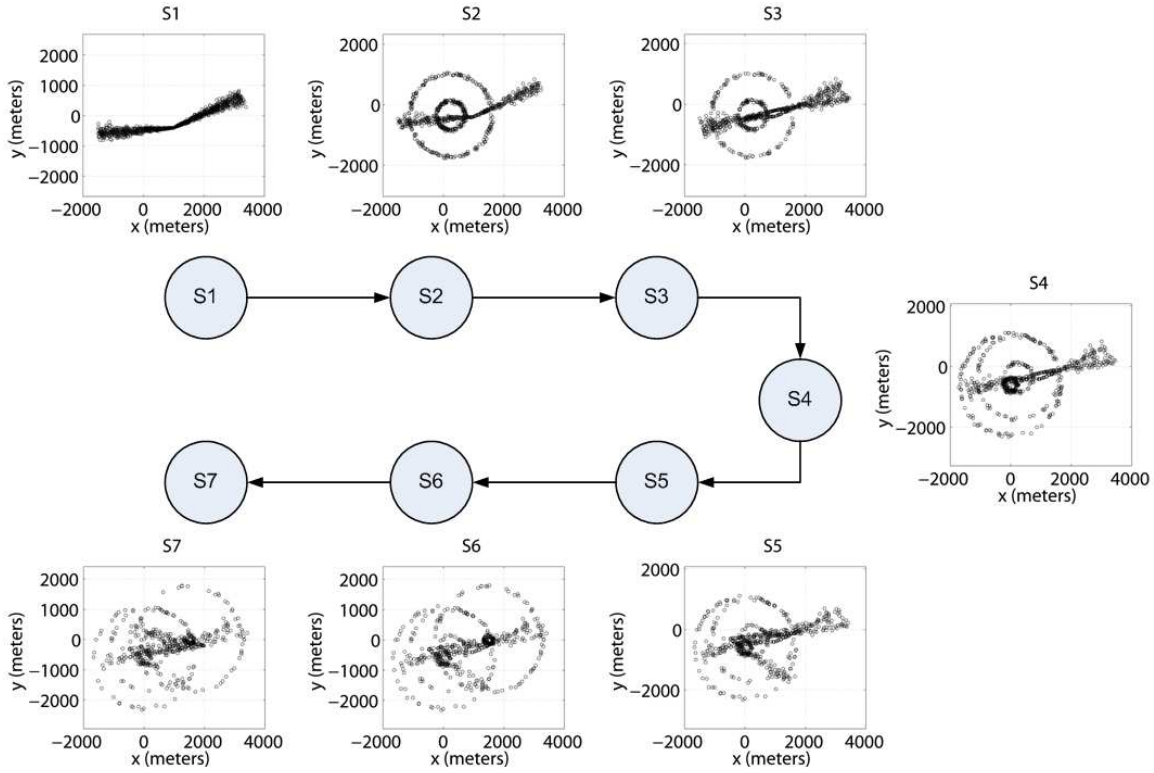
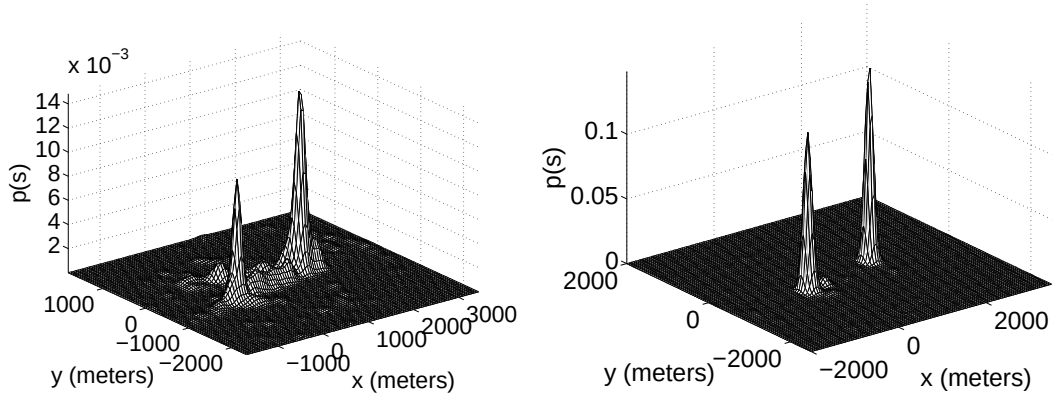


Figure 36. Particle evolution during the downward pass through a communication chain using our initialization algorithm.



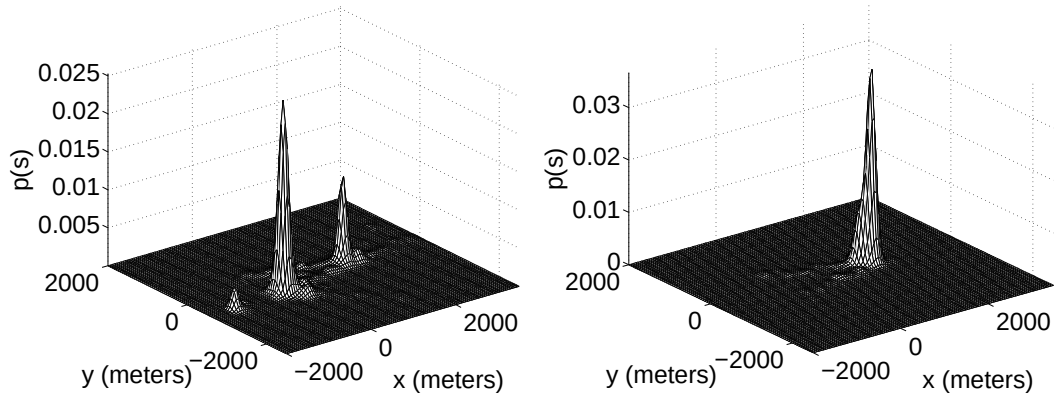
(a) Our algorithm. (b) BP implementation.

Figure 37. Comparing the posterior distributions generated by our initialization algorithm and the BP implementation when all sensors detect all targets. The chain communication topology is used.

4.3.2.2 Simulation II

Although the BP implementation showed good performance in simulation I, we expect performance to drop in the presence of target occlusions. In this simulation, only S1 and S7 see both targets. S2, S4 and S5 only see target 1 whereas S3 and S6 only see target 2. The

same chain communication topology given in Figure 34 is used. The final posterior distributions generated by our initialization algorithm and the BP implementation are given in Figure 38. It can be seen that even though our initialization algorithm may generate a pos-



(a) Our algorithm. (b) BP implementation.

Figure 38. Comparing the posterior distributions generated by our initialization algorithm and the BP implementation in the presence of target occlusions. The chain communication topology is used.

Table 2. Comparing detection rates for our initialization algorithm and the BP implementation in the presence of target occlusions. The chain communication topology is used.

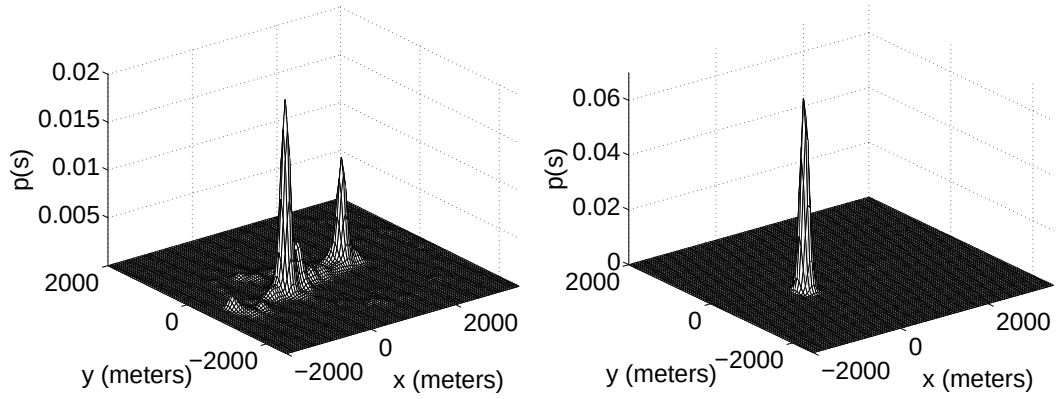
Iteration, Detections	Our Algorithm Target 1	Our Algorithm Target 2	BP Target 1	BP Target 2
1	1	1	1	0
2	1	1	0	1
3	1	1	0	1
4	1	1	0	1
5	1	1	1	0
6	1	1	1	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	0
10	1	1	0	1
Detection Rate (100 runs)	0.99	0.96	0.43	0.59

terior distribution with minor anomalies in the tails, strong peaks appear at the true target states. However, the BP implementation ends up losing one of the two targets completely.

We have repeated this simulation 100 times with different random number realizations. The detected targets over the first 10 iterations along with the detection rates over all 100 iterations are given in Table 2. It can be seen that our initialization algorithm significantly outperforms the BP implementation.

4.3.2.3 Simulation III

We repeat the experiment from simulation II by reversing the order of the nodes in the communication chain. Therefore, in this simulation, S7 launches the initialization algorithm, and the posterior distribution evolves as it passes through the communication chain towards S1. The same occlusions from simulation II are repeated. The final posterior distributions generated by our initialization algorithm and the BP implementation are given in Figure 39. Just as in the previous simulation, the BP implementation once again loses one target completely, whereas our initialization algorithm produces a posterior distribution with peaks at the true target locations. We have repeated this simulation 100 times with different random number realizations. The detected targets over the first 10 iterations along with the detection rates over all 100 iterations are given in Table 3. Once again, it is clear that our initialization algorithm outperforms the BP implementation.



(a) Our algorithm. (b) BP implementation.

Figure 39. Comparing the posterior distributions generated by our initialization algorithm and the BP implementation in the presence of target occlusions. The chain communication topology is used in reverse order.

Table 3. Comparing detection rates for our initialization algorithm and the BP implementation in the presence of target occlusions. The chain communication topology is used in reverse order.

Iteration, Detections	Our Algorithm Target 1	Our Algorithm Target 2	BP Target 1	BP Target 2
1	1	0	1	0
2	1	1	1	0
3	1	1	0	1
4	1	1	1	0
5	1	1	0	1
6	1	1	1	1
7	1	1	1	0
8	1	1	0	1
9	1	1	0	1
10	1	0	1	0
Detection Rate (100 runs)	0.98	0.99	0.5	0.59

4.3.2.4 Simulation IV

In this simulation, we compare the performance of our initialization algorithm with that of the BP implementation in a sensor network using the tree communication topology given in Figure 40. Occlusions are simulated as follows: S1, S4, S5, S6 and S7 detect both

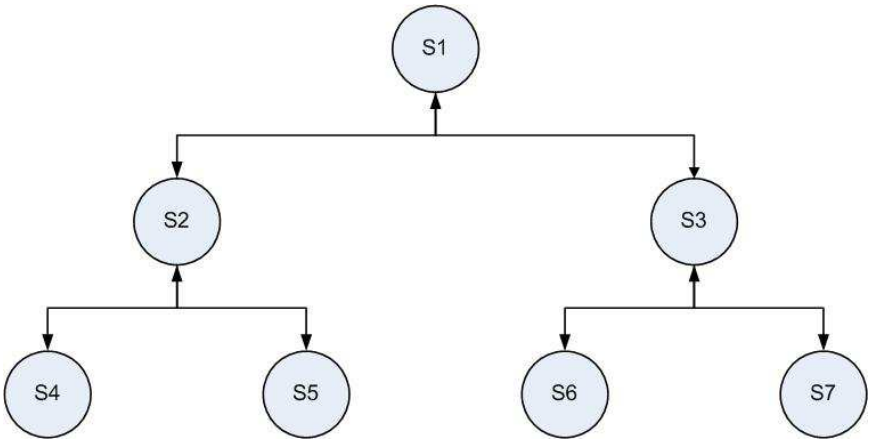
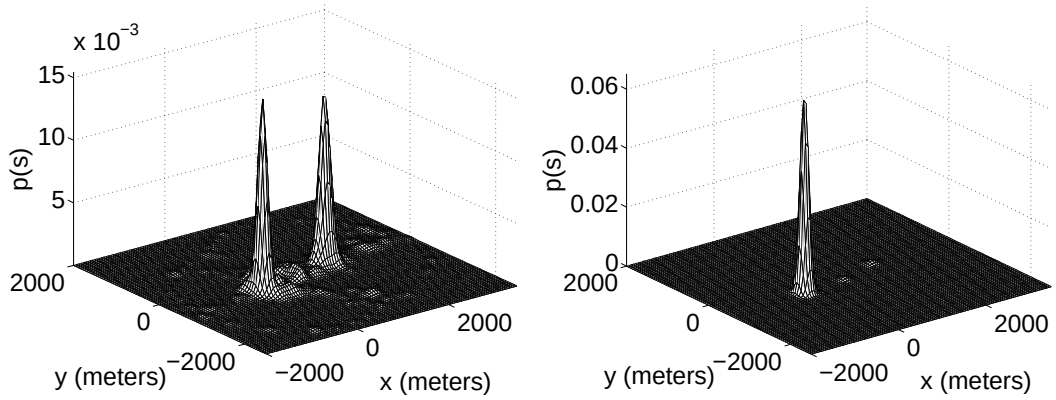


Figure 40. Communication tree for the simulated network

targets, S2 only detects target 1, and S3 only detects target 2. The final posterior distributions generated by our initialization algorithm and the BP implementation are given in

Figure 41. Consistent with the results of previous simulations, it can be seen that even



(a) Our algorithm. (b) BP implementation.

Figure 41. Comparing the posterior distributions generated by our initialization algorithm and the BP implementation in the presence of target occlusions. The tree communication topology is used.

Table 4. Comparing detection rates for our initialization algorithm and the BP implementation in the presence of target occlusions. The tree communication topology is used.

Iteration, Detections	Our Algorithm Target 1	Our Algorithm Target 2	BP Target 1	BP Target 2
1	1	1	1	1
2	1	1	1	0
3	1	1	1	0
4	1	1	1	1
5	1	1	1	0
6	1	1	1	0
7	1	1	1	0
8	1	1	1	1
9	1	1	0	1
10	1	1	1	0
Detection Rate (100 runs)	1	1	0.97	0.44

though our initialization algorithm generates a posterior distribution with small bumps in the low probability regions, dominant peaks appear at the true target states. However, the BP implementation once again loses one target completely. We have repeated this simulation 100 times with different random number realizations. The detected targets over the

first 10 iterations and the detection rates over all 100 iterations are given in Table 4. Consistent with previous results, out initialization algorithm significantly outperforms the BP implementation.

4.4 Summary

In this chapter, we have compared our distributed initialization algorithm with nonparametric implementations of BP. A fully exhaustive comparison on arbitrary networks and target configurations is not feasible. Therefore, we have simulated the BP implementation under the same network conditions as our initialization algorithm to ensure a fair comparison. With an infinite number of particles, both algorithms are expected to perform similarly. However, we do not have the luxury of using infinite particles. Both algorithms use importance sampling, but with different importance functions. From the theoretical development and simulation results presented in this chapter, it is clear that both implementations have advantages and disadvantages. The primary advantage of the BP implementation is that the final joint posterior distribution is generated by particles concentrated in high probability regions of the target state space. The resulting particle distribution provides high resolution at the modes of the distribution. Thus, the BP implementation is capable of generating accurate estimates using fewer particles than our initialization algorithm. This is particularly true in large networks in which the individual sensor nodes lack local observability. However, the BP implementation suffers from a lack of robustness compared to our initialization algorithm. The BP implementation showed inconsistent and poor performance in our simulations when some targets were occluded at some sensors, and when the communication topology and scheduling changed. Therefore, we recommend using the BP implementation for initializing sensor networks with accurate and reliable local estimates, and using our initialization algorithm in networks with unreliable local estimates.

CHAPTER 5

IMPROVING THE EFFICIENCY OF KERNEL DENSITY ESTIMATION

5.1 Introduction

Nonparametric density estimation methods like KDE offer advantages over parametric methods in that no rigid assumptions are made about the underlying density from which observations are generated. However, these advantages come at the cost of a significant increase in computation. In the low complexity initialization algorithm of Section 2.2, KDE was not used. This was possible because the formulation of the algorithm allowed for the evolving posterior density to be modified sequentially at each node based on the organic state estimates at the current node. This algorithm, though computationally efficient, came at the cost of increased communication through the network, requiring 3 communication passes even with a simple chain communication topology for global initialization. Another drawback was that the algorithm, in its basic form, did not find direct applicability to sensor networks with arbitrary communication topologies. To address these issues, modifications were made to the initialization algorithm.

Consider the low latency initialization algorithm given in Section 2.3 and the generalized initialization algorithm in Section 3.3. Both of these algorithms use KDE at each node to estimate posterior densities represented by the received particles and importance weights. These estimated densities are then used (i) to assign new importance weights to different sets of particles based on organic state estimates that are not directly available at the current node, but are contained in the density represented by some set of received particles and importance weights, and (ii) to modify the importance weights to account for the constantly varying particle support. The naive exhaustive method currently in use calculates the distances between each particle in one set and each particle in another set individually. Parametric methods of density estimation, though computationally more efficient, are not

used because the resulting posterior distributions typically can not be represented using standard distributions.

For convenience, we restate the equations that dominate the computational load of the generalized initialization algorithm for tree communication given in Chapter 3. In the downward pass, each node receives a density from its parent and fuses the received density with its own local posterior. If D particles are used to represent the evolving posterior distribution, computing the scaled weights requires $O(D^2)$ operations, and kernel evaluations, at each node and has the form

$$\tilde{w}_t^{(i)} \propto p\left(\mathbf{s}_t^{(i)} | \mathbf{z}_{m,t}\right) \cdot \sum_{j=1}^{D} w_{r,t}^{(j)} W\left(\mathbf{s}_t^{(i)} - \mathbf{s}_{r,t}^{(j)}\right). \tag{87}$$

In the upward pass, each parent node receives a density from each of its N children. The job of the parent is to effectively fuse these received densities together. Computing the scaled weights for each particle at each parent node in the upward pass requires $O(ND)$ operations, and kernel evaluations, and has the form

$$\tilde{w}_t^{(i)} = \frac{\prod_{n=1}^{N}\left(\sum_{j=1}^{D} w_{\mathrm{C}_{l,n},t}^{(j)} W\left(\mathbf{s}_t^{(i)} - \mathbf{s}_{\mathrm{C}_{l,n},t}^{(j)}\right)\right)}{\left(\sum_{j=1}^{D} \hat{w}_{\mathrm{P}_l,t}^{(j)} W\left(\mathbf{s}_t^{(i)} - \hat{\mathbf{s}}_{\mathrm{P}_l,t}^{(j)}\right)\right)^{N-1}}. \tag{88}$$

Since there are a total of ND particles for which scaled weights must be computed, the total computational load at each parent node is $O(N^2D^2)$. For typical values of D and N, the computational power required to evaluate (87) and (88) can slow down the initialization algorithm.

We focus on three avenues for reducing computation. The first is the choice of kernel function. Some kernel functions can be evaluated using fewer computations than others. Since the upward pass, implemented in the naive form, requires $O(N^2D^2)$ evaluations, every mathematical operation that is eliminated could reduce the overall processing time significantly. The second is a restructuring of the algorithm in which we reverse the order of the weighting and resampling steps at each node. This restructuring reduces the number of

particles before computing weights using KDE. The third avenue for computation reduction is partitioning the data. Note that the argument of $W(\cdot)$ in (87) and (88) is the distance between a pair of particles. Depending on the choice of kernel function, the contribution to the weight of a particle from particles that lie far from it in the state space could be negligible, or even zero. It would be advantageous to partition the sets of particles according to the geometry of the particle distribution so chunks of comparisons could be eliminated, hence reducing the number of operations needed.

The choice of kernel bandwidth, or smoothing parameter, defines the size of the kernel function and can have a significant impact on both the accuracy and the computational load of KDE. Because the Gaussian kernel has infinite support, the choice of bandwidth does not affect the computational load. The Epanechnikov kernel and the spherical kernel, on the other hand, have finite support. This can be exploited to avoid evaluating the contribution to the kernel density estimate at a point from particles that lie outside the region of support. Since the kernel bandwidth directly affects the region of support, it has a direct impact on the computational load of KDE. Hence the bandwidth for the Epanechnikov kernel and the spherical kernel must be appropriately selected.

There are various methods to select the kernel bandwidth. The most popular methods use a form of leave-one-out cross validation [47]. These methods determine cross validation scores for a set of B bandwidths under consideration, and select the bandwidth that gives the lowest score. These cross validation methods are computationally expensive and only provide the optimal bandwidth if it is contained within the set of bandwidths under consideration. Therefore, we avoid using cross validation. Instead, we choose the bandwidth subjectively such that the resulting density estimates are smooth plots [24].

It is shown in [24] that given the optimal kernel bandwidth, the choice of the particular kernel function has a negligible impact on the accuracy of the estimated density. Hence, when comparing across kernel functions, we focus primarily on the computational

demands. Measures of accuracy are used to compare across the three methods for computation reduction, and are valid only when other factors such as the number of particles, and kernel function and bandwidth, are kept constant.

5.2 Choice of Kernel Function

In theory, any kernel function can be used for KDE. However, we focus on kernels that are probability densities themselves to ensure that the resulting kernel density estimate is a pdf as well [24]. We further restrict our focus to kernels that are unimodal and radially symmetric because this eliminates the effect of orientation. Three commonly used kernels satisfying the above mentioned properties will be discussed and compared in this section.

5.2.1 Gaussian Kernel

The Gaussian kernel is the most commonly used kernel function for KDE. The functional form of the multivariate Gaussian kernel is given by

$$K_{\mathrm{G}}(\mathbf{x},\mathbf{y}) = \frac{1}{(2\pi)^{d/2}\,|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{y})^{\mathrm{T}}\Sigma^{-1}(\mathbf{x}-\mathbf{y})\right), \tag{89}$$

where d is the dimensionality of the state space and Σ is the covariance matrix. The standard Gaussian kernel is plotted in 1 and 2 dimensions in Figure 42.

The Gaussian kernel is a popular choice in KDE due to its continuity and differentiability properties. However, it is expected to have a higher computational load than the other kernels discussed below for two reasons: (i) evaluating the exponential function during each call to the kernel in (89) is cumbersome and (ii) the infinite support does not allow pruning data without introducing roundoff errors.

5.2.2 Epanechnikov Kernel

Another commonly used kernel function is the Epanechnikov kernel. The functional form of the Epanechnikov kernel is given by

$$\begin{aligned} K_{\mathrm{E}}(\mathbf{x},\mathbf{y}) &= \tfrac{d+2}{2V_{d,h}}\left(1-\tfrac{\|\mathbf{x}-\mathbf{y}\|^2}{h^2}\right) &&, \|\mathbf{x}-\mathbf{y}\| \le h \\ K_{\mathrm{E}}(\mathbf{x},\mathbf{y}) &= 0 &&, \|\mathbf{x}-\mathbf{y}\| > h \end{aligned}, \tag{90}$$

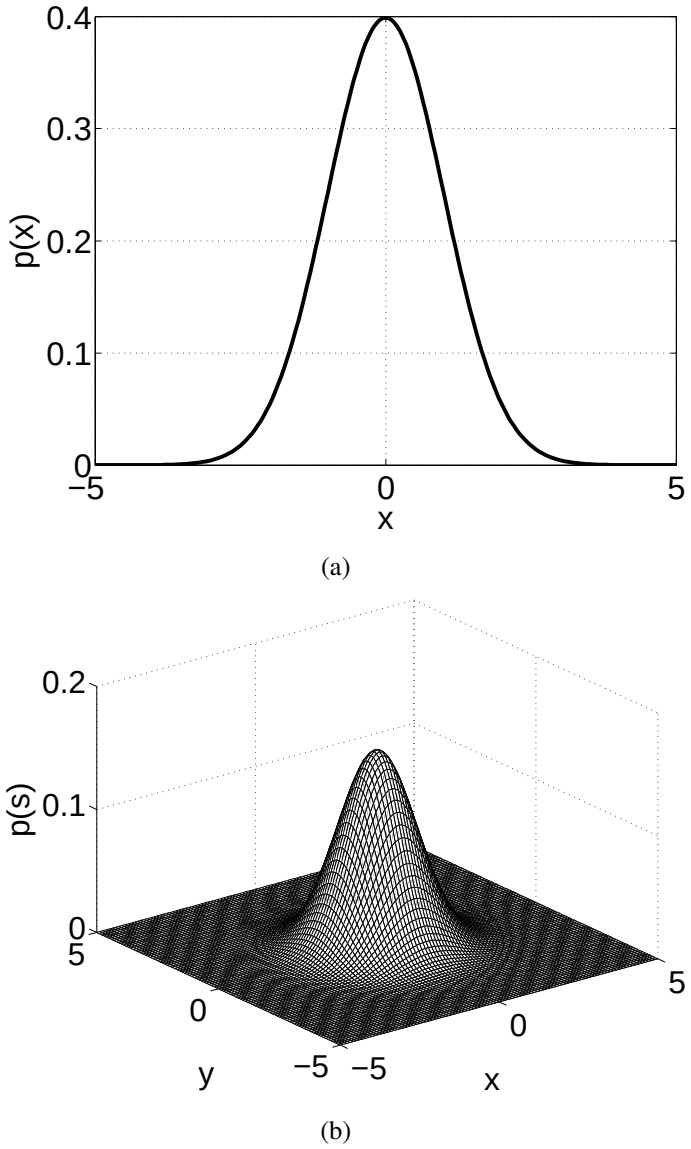


Figure 42. Gaussian kernel functions with unit bandwidth. (a) 1-d. (b) 2-d.

where h is the kernel bandwidth, or smoothing parameter, and $V_{d,h}$ is the volume of the d dimensional hypersphere with radius h. The Epanechnikov kernel with unit bandwidth is plotted in 1 and 2 dimensions in Figure 43.

The Epanechnikov kernel was first used for density estimation in [48], where it was shown, asymptotically, to be the most efficient among all possible kernels at minimizing the mean integrated square error (MISE), given the optimal bandwidth. MISE is defined as

$$\text{MISE} = E \int \left(f(\mathbf{s}) - \hat{f}(\mathbf{s}) \right)^2 d\mathbf{s}, \tag{91}$$

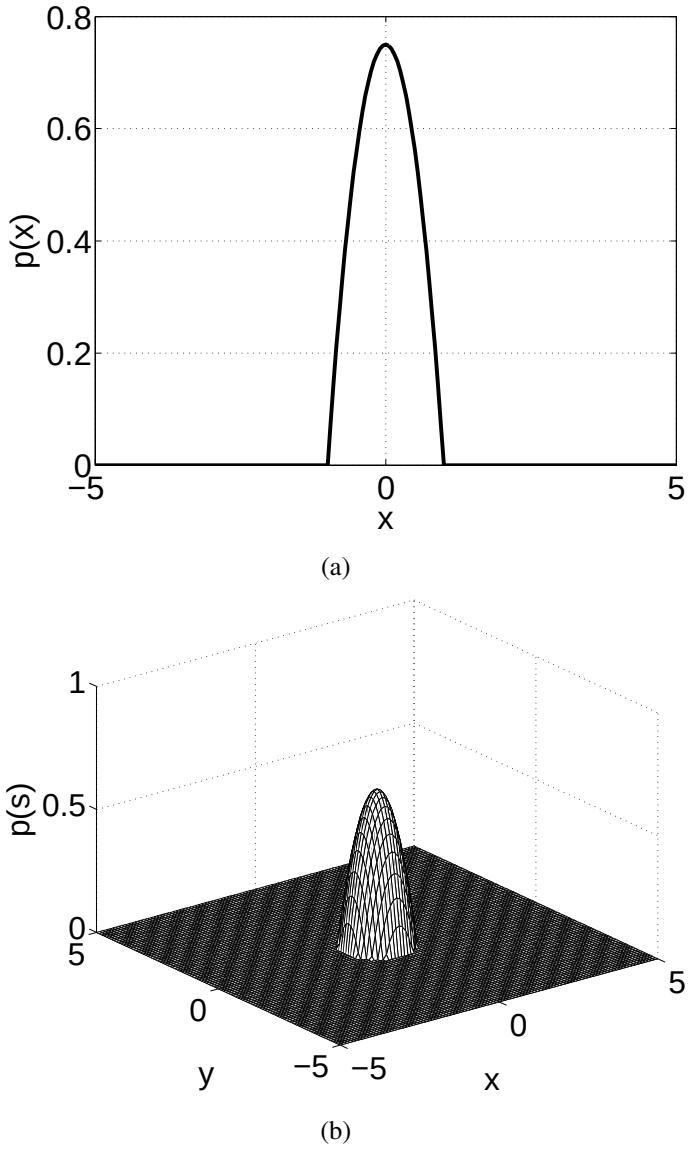


Figure 43. Epanechnikov kernel functions with unit bandwidth. (a) 1-d. (b) 2-d.

where $f(\mathbf{s})$ and $\hat{f}(\mathbf{s})$ represent the true density and the estimated density, respectively. This optimal efficiency makes the Epanechnikov kernel an attractive choice for KDE. When comparing (90) and (89), it can be seen that the Epanechnikov kernel requires fewer computations than the Gaussian kernel. Also the Epanechnikov kernel has finite support, which allows pruning data to further reduce the computational load. However, the Epanechnikov kernel does not have the nice higher order differentiability properties of the Gaussian kernel. Since we are interested in finding the peaks, higher order differentiability is not a

matter of concern in our application.

5.2.3 Spherical Kernel

The functional form of the spherical kernel is given by

$$\begin{aligned} K_{\mathrm{S}}(\mathbf{x},\mathbf{y}) &= \tfrac{1}{V_{d,h}} \quad , \|\mathbf{x}-\mathbf{y}\| \leq h \\ K_{\mathrm{S}}(\mathbf{x},\mathbf{y}) &= 0 \quad , \|\mathbf{x}-\mathbf{y}\| > h \end{aligned}, \tag{92}$$

where $V_{d,h}$ is the volume of the d dimensional hypersphere with radius h and, with unit bandwidth, is plotted in 1 and 2 dimensions in Figure 44.

Comparing (92) to (90) and (89), it can be seen that the spherical kernel is the simplest among the kernel functions considered and requires the least number of computations for evaluation. It also shares the property of finite support with the Epanechnikov kernel, allowing data to be pruned to further reduce computation. However, it can introduce sharp discontinuities in the estimated density, especially if small data sets are used. Hence, care must be taken when using the spherical kernel for KDE as it may cause problems in certain applications.

5.2.4 Comparing the Different Kernels

We repeat the simulation of Section 3.3.2.1 with varying kernel functions. For each choice of kernel, simulations are repeated with a varying number of particles. The total amount of time spent by the initialization algorithm in the KDE function is recorded using MATLAB's Profiler utility and the average time over 10 runs is presented in Table 5. The average position and velocity estimation errors, calculated as the average Euclidean distance between the estimated target states and the true target states over 10 runs, are given in Tables 6 and 7. As expected, the estimation error increases as the number of particles decreases. However, for the same number of particles, the estimation error is comparable for the different kernel functions. The final density estimates, generated using 1000 particles, are plotted in Figure 45. The three plots, though constructed using different kernel functions and bandwidths, look very similar to each other. Therefore, we claim the bandwidths for the different kernels

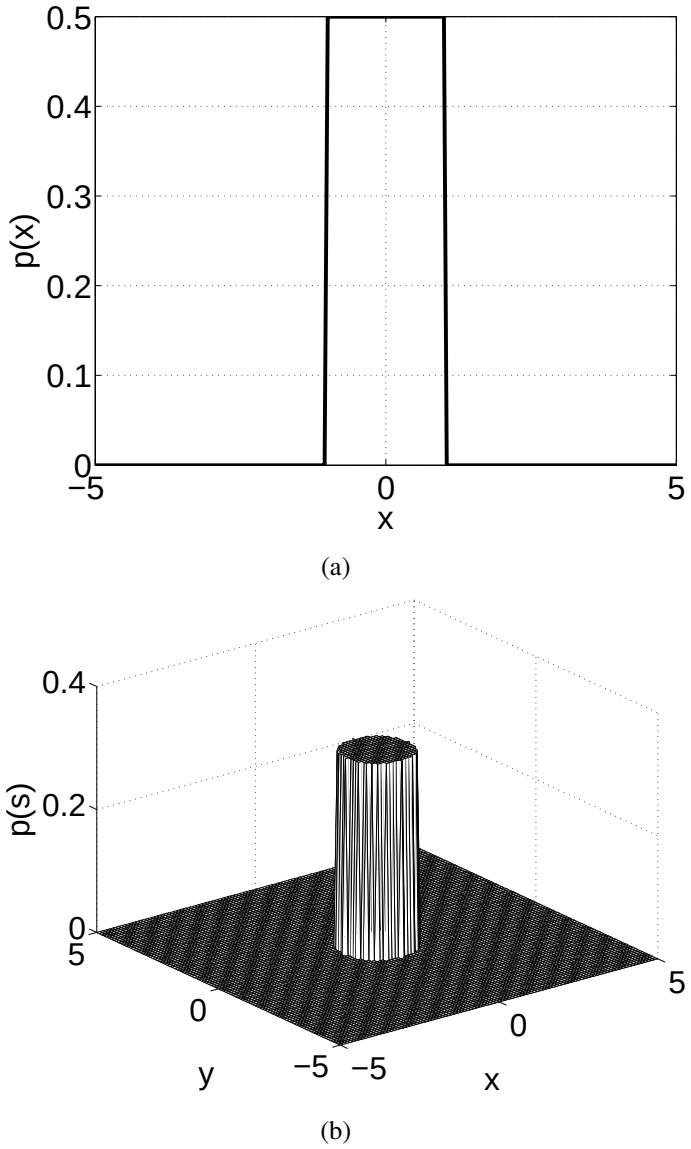


Figure 44. Spherical kernel functions with unit bandwidth. (a) 1-d. (b) 2-d.

are appropriately selected and fair comparisons of computational demands can be made.

As the number of particles increases, the computational load also increases, as seen in Table 5. As the number of particles is doubled, the computational load increases by a factor of about 8. This behavior is seen going from 200 particles to 400 particles, and from 400 particles to 800 particles. From (87) and (88), the downward pass requires $O(D^2)$ operations, whereas the upward pass requires $O(N^2D^2)$ operations. The communication tree used

Table 5. Comparing computational time, in seconds, for KDE using different kernel functions. Results are the average of 10 runs.

Kernel, Particles	200	400	600	800	1000
Gaussian	11.476	88.043	290.82	684.068	1336.055
Epanechnikov	11.45	87.761	286.633	669.832	1313.903
Spherical	10.896	86.681	284.999	664.901	1301.085

Table 6. Comparing average position estimation error, in meters, for KDE using different kernel functions. Results are the average of 10 runs.

Kernel, Particles	200	400	600	800	1000
Gaussian	73.776	48.761	38.338	32.422	25.209
Epanechnikov	73.734	47.664	39.469	34.901	23.769
Spherical	74.476	48.034	39.385	33.573	25.276

in this simulation has a binary structure in which each parent node has two children. Therefore, doubling the number of particles leads to a fourfold increase in computational time for the downward pass, and a sixteenfold increase in computational time for the upward pass. The net increase in computational time for the algorithm with 2 times the number of particles is therefore expected to be between 4 times and 16 times, and happens to be around 8 times in this simulation.

Comparing across kernel functions, it can be seen that KDE requires similar computational time regardless of the particular choice of kernel. Therefore, when using the naive

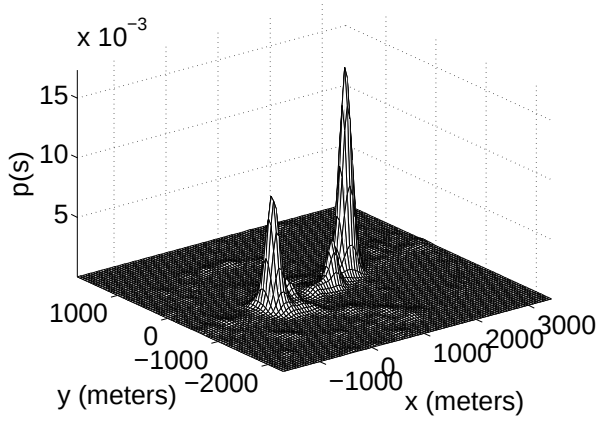
Table 7. Comparing average velocity estimation error, in meters/second, for KDE using different kernel functions. Results are the average of 10 runs.

Kernel, Particles	200	400	600	800	1000
Gaussian	6.398	5.542	4.906	4.636	4.389
Epanechnikov	6.491	5.352	5.19	4.554	4.012
Spherical	6.539	5.409	5.003	4.743	4.538

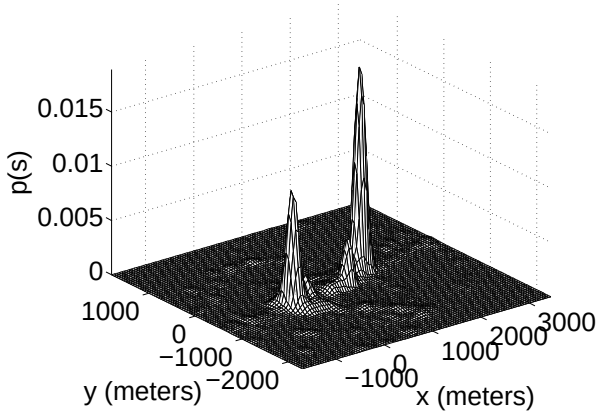
exhaustive method, the choice of kernel should be driven by factors other than computational demands. As expected, the spherical kernel, being the simplest, offers the minimum computational load among the kernels considered. However, the spherical kernel has sharp discontinuities and the reduction in computational load offered over the Epanechnikov kernel is marginal. The asymptotically optimal efficiency and continuity properties of the Epanechnikov kernel make it an attractive kernel for KDE if the higher-order differentiability properties of the Gaussian kernel are not required.

5.3 Restructuring the Algorithm

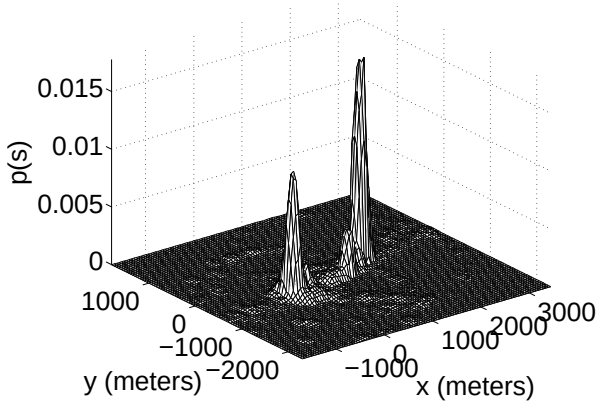
The initialization algorithms in Chapters 2 and 3 followed four basic steps: (i) receive particles and importance weights representing certain joint densities, (ii) use KDE to generate scaled weights for each received particle using *all* received particles and importance weights, (iii) resample to ensure the communication bandwidth remains fixed and the particle distribution is an equally weighted mixture of individual node posteriors, and (iv) modify the scaled weights based on the final particle support to generate the final importance weights. Typically, the computational load is dominated by step (ii). A method to reduce the overall computational load would be to switch the order of steps (ii) and (iii). According to the proposed modification, we first resample particles to represent an equally



(a)



(b)



(c)

Figure 45. KDE using different kernel functions: (a) Gaussian kernel (b) Epanechnikov kernel (c) Spherical kernel.

weighted mixture of individual posteriors and use this reduced particle set to assign scaled

weights. This modification brings about some important changes in the initialization algorithms for chain and tree communication as discussed below.

Consider the forward pass of the low complexity initialization algorithm for chain communication given in Section 2.2. Let D represent both the number of particles and importance weights received from the preceding node in the chain, and the number of newly generated particles at the current node. Using the proposed modification, let D_r and D_n represent the number of received particles and newly generated particles, respectively, that survive the resampling stage. After resampling, the importance weights for the surviving received particles are normalized to sum to unity. This is necessary to ensure that the received particles and importance weights still represent a density. The total number of particles for which scaled weights have to be computed now reduces from $2D$ to $D = D_r + D_n$. With this reduction in the number of particles, the computational load at each node reduces from $O(D^2)$ to $O(D_r D)$.

Following the resampling algorithm in Section 2.2.1.3, it can be shown that with each hop in the communication chain, $E\{D_r\}$ increases while $E\{D_n\}$ decreases. This is because the number of individual node posteriors contained within the joint posterior represented by the received set of D particles and importance weights keeps increasing with each hop in the communication chain, whereas the D newly generated particles represent the local posterior for the current node only. Hence, greater savings in computation can be achieved during the first few hops in the communication chain.

The computational analysis for the low complexity initialization algorithm for chain communication given above is also applicable to the downward pass of the generalized initialization algorithm given in Section 3.3. Now consider the upward pass of the generalized initialization algorithm. Let D represent the number of particles and importance weights received at a parent node from each child node. Using the proposed modification, let D_n represent the number of particles received from the n^{th} child node that survive the resampling stage, with $n = 1, \ldots, N$. The D_n importance weights corresponding to these

surviving particles must be modified so they, along with the D_n particles, represent the same density that the full set of D particles and importance weights received from that child node represented. Recall from the resampling stage of Section 3.3.1.2, each set of particles received from a child node was separated into two distinct sets: a set of particles representing successor posteriors only and a set of particles representing ancestor posteriors only. The ancestor posteriors, being redundantly represented by the received particles from each of the N child nodes, were weighted $1/N$ times as much as the successor posteriors when resampled. This step was taken to ensure that the final combined set of D particles represents an equally weighted mixture of individual posteriors, without over-representing the ancestor posteriors. However, each set of D_n received particles that survives resampling, considered independently, now has ancestral posteriors under-represented. This discrepancy can be corrected by modifying the importance weights of the surviving particles. The particles representing the successor posteriors maintain their importance weights unchanged. The particles representing the ancestor posteriors have their importance weights multiplied by N to compensate for the $1/N$ weighting assigned to them when resampled. The resulting D_n importance weights are normalized to sum to unity. This normalization step is necessary to ensure that the particles and importance weights together represent a density. This procedure is repeated for each of the N sets of received particles and importance weights that survive resampling. The new particles and importance weights are used to evaluate scaled weights. The total number of particles at which scaled weights must be evaluated at a parent node now reduces from ND to $D = \sum_{n=1}^{N} D_n$. With the reduction in the number of particles, the computational load of the algorithm reduces from $O(N^2D^2)$ to $O(D^2)$ at each node.

The simulation from Section 3.3.2.1 is repeated with the restructured algorithm proposed in this section. The time spent within the KDE function is recorded using MATLAB's Profiler utility and the average time over 10 runs is shown in Table 8 for different

sizes of particle sets. For convenience, the time spent by the original generalized initialization algorithm of Chapter 3 in the KDE function is re-tabulated from Table 5. A Gaussian kernel function is used with both algorithms. It can be seen that significant savings in computation are possible if resampling is performed before the weighting stage.

Table 8. Comparing computational time, in seconds, for KDE using different algorithmic structures. Results are the average of 10 runs.

Algorithm, Particles	200	400	600	800	1000
Resampling after Weighting	11.476	88.043	290.82	684.068	1336.055
Resampling before Weighting	5.372	36.7835	120.751	278.383	537.195

One might say that with the savings in computation offered by the modification proposed in this section, why would one ever consider using the original algorithms presented in Chapters 2 and 3. It is a well known fact in both importance sampling and density estimation that, other factors kept constant, the quality of estimates improves with the number of particles. As the number of particles reduces, information can only be lost [43]. To maximize accuracy, the full sets of received particles and importance weights were used to assign scaled weights in the original algorithms. Even though the modifications in this section show promising results for reducing computation, a loss in accuracy is expected. Since each parent node in the communication tree used in our simulation has 2 children, about $1/2$ as many particles are used to evaluate scaled weights in the restructured algorithm compared to the original algorithm. Therefore the estimation accuracy provided by the restructured algorithm using D particles is expected to be in the ballpark of that provided by the original algorithm using $D/2$ particles. This is shown in Tables 9 and 10.

Table 9. Comparing average position estimation error, in meters, for KDE using different algorithmic structures. Results are the average of 10 runs.

Algorithm, Particles	200	400	600	800	1000
Resampling after Weighting	73.776	48.761	38.338	32.422	25.209
Resampling before Weighting	155.859	73.586	61.203	50.699	43.365

Table 10. Comparing average velocity estimation error, in meters/second, for KDE using different algorithmic structures. Results are the average of 10 runs.

Algorithm, Particles	200	400	600	800	1000
Resampling after Weighting	6.398	5.542	4.906	4.636	4.389
Resampling before Weighting	9.859	6.586	6.033	5.625	5.279

5.4 Partitioning Data Using Kd-Trees

So far, every implementation of KDE that has appeared in this dissertation has used a naive exhaustive method in which the distances between each point in one set and each point in another set are calculated individually as given in (87) and (88). In this section, we look at space partitioning data structures called k-dimensional trees, or *kd-trees* [49], that help us reduce the total number of comparisons needed for KDE by exploiting the geometry of the data sets. In a kd-tree, data is hierarchically organized in a binary tree structure. Each node holds a set of data points contained within a hyperrectangular bounding box that is maximally tight along each coordinate axis. Each non-leaf node has two child nodes. The hyperrectangular bounding box of the parent node is split along the coordinate axis with the largest variance such that half of the data points lie on either side of the splitting hyperplane. The resulting sets of data points are assigned to distinct child nodes and new

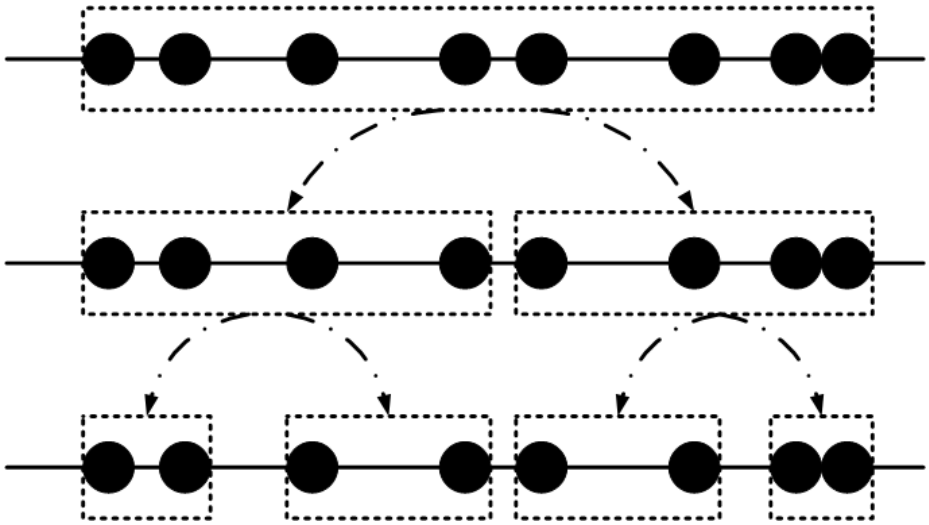


Figure 46. Sample one dimensional kd-tree. • represent data points contained within bounding boxes.

hyperrectangular bounding boxes are defined accordingly. The splitting process continues until each node contains only one data point. Typically, the root node contains the entire set of data points. A sample kd-tree is shown in Figure 46. In addition to the data points, kd-trees may also store local statistics at each node. Kd-trees containing this additional information are known as multi-resolution kd-trees, or *mrkd-trees*[50].

A detailed discussion on the improved efficiency of KDE with the use of kd-trees is given in [51]. For convenience, we will briefly review some important points. Let us consider two sets of D particles along with their associated importance weights: a query set $\mathcal{S}_q$ and a density set $\mathcal{S}_d$. The kernel density estimate is generated using the particles and weights contained in $\mathcal{S}_d$. We wish to evaluate the density at all points contained within $\mathcal{S}_q$. Let us consider first the single-tree algorithm in which a kd-tree T_d partitions the data in $\mathcal{S}_d$ and each query point $s_q \in \mathcal{S}_q$ is considered sequentially. T_d is traversed in a depth-first manner. At each node in the tree, upper and lower bounds on the contribution to the density at s_q from any particle in the current node can be computed using the hyperrectangular bounding box for that node, as shown in Figure 47. If the difference between these bounds is less than a preselected threshold δ, we can use a constant-mass centroid approximation by which we can prune that node, approximating its contribution to the density estimate at s_q

by the mass of its centroid. As $\delta \rightarrow 0$, this method can be used to prune nodes that contain only those particles making equal contribution to the density at s_q. If a node is pruned, there is no necessity to recurse on its children. Depending on the choice of kernel and threshold, pruning could give exact results, especially for the case of finite extent kernels like the spherical kernel and the Epanechnikov kernel. However, in the case of infinite extent kernels like the Gaussian kernel, pruning typically produces approximate results unless δ is small enough to be comparable to machine precision. Regardless, pruning offers significant savings in computation because the contributions to the density estimate from chunks of data can be estimated simultaneously. Tree construction requires $O(D\log D)$ operations and is needed only once for the life of the data set. The single-tree algorithm requires $O(D\log D)$ operations, a significant reduction from $O(D^2)$ required by the naive exhaustive method. The benefits of kd-trees can be further amplified by using the dual-tree algorithm, in which the query particles are also organized into a separate kd-tree T_q. The improved efficiency is achieved by comparing chunks of query particles to chunks of density particles as shown in Figure 48, instead of comparing single query particles to chunks of density particles as in the single-tree algorithm. The dual-tree algorithm scales as $O(D)$.

It has been pointed out, in Chapters 2 and 3, that KDE dominates the computational load of the initialization algorithm. This is particularly true when assigning scaled weights. We can reap the benefits of kd-trees and reduce the computational load by first organizing separate sets of particles and importance weights into separate kd-trees, and then evaluating densities using dual-tree recursion on pairs of kd-trees taken in sequence. Note that since the particle sets representing the joint posteriors are constantly changing, new kd-trees have to be constructed at each node. This adds to the computational overhead and will be included in our analysis. The code used to generate kd-trees and to perform the dual-tree recursions can be found in [52].

The simulation from Section 3.3.2.1 is repeated with the modifications proposed in this

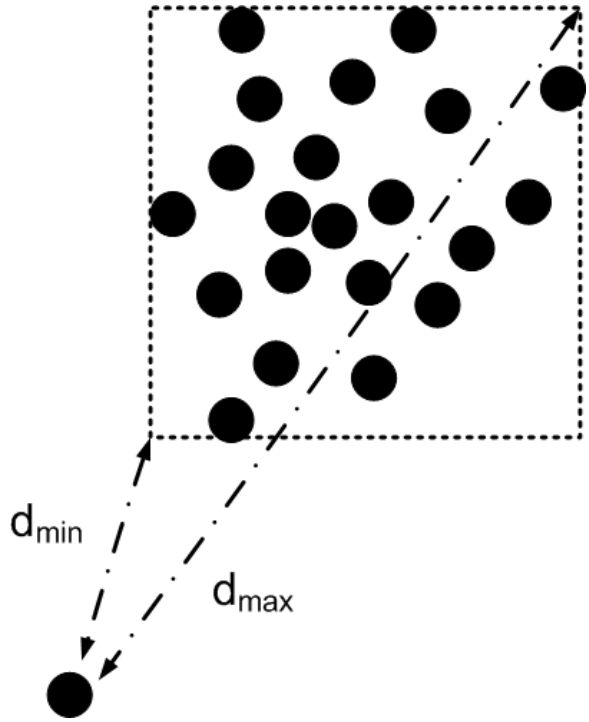


Figure 47. Single-tree algorithm using point-node comparison.

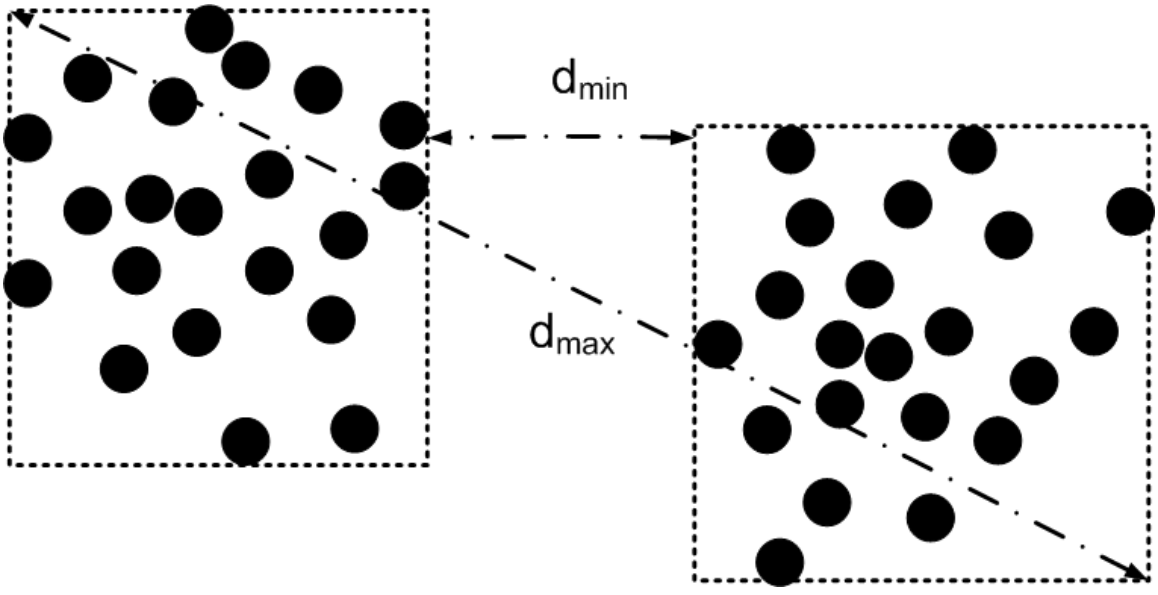


Figure 48. Dual-tree algorithm using node-node comparison.

Table 11. Computational time in seconds for KDE using kd-trees. Results are the average of 10 runs.

Algorithm, Particles	200	400	600	800	1000
Naive, Gaussian	11.476	88.043	290.82	684.068	1336.055
Naive, Epanechnikov	11.45	87.761	286.633	669.832	1313.903
kd-trees, Gaussian	0.633	1.618	3.39	4.986	7.023
kd-trees, Epanechnikov	0.055	0.091	0.196	0.263	0.388

section. The time spent within the KDE function is recorded using MATLAB's Profiler utility and shown in Table 11 for different sizes of particle sets. For convenience, the time spent by the original generalized initialization algorithm of Chapter 3 in the KDE function is re-tabulated from Table 5. It must be noted that some of the code implementing kd-trees uses MEX functions along with MATLAB code, whereas the naive exhaustive method uses purely MATLAB code. Therefore the savings in computation may be somewhat exaggerated. However, our results show savings similar in order to those given in [51]. A few important points stand out in this table. First, the use of kd-trees offers significant savings in computation over the naive method for KDE. Second, even though the differences in the savings in computation offered by the choice of kernel function were marginal when using the naive exhaustive method, the same choice can make a significant difference when kd-trees are used. It can be seen that the Epanechnikov kernel is notably more efficient than the Gaussian kernel. This is expected because the Epanechnikov kernel, having finite extent, when used for KDE, need only be evaluated for points that lie within the region of support of the kernel to provide exact results. This feature of finite extent kernels offers improved pruning capacity compared to the Gaussian kernel, which, on the other hand,

must be evaluated far into the tails to minimize approximation errors.

Because kd-trees improve the efficiency of our algorithms by exploiting the geometry of the particle distribution, no loss in accuracy is expected for sufficiently small values of δ. We use $\delta = 10^{-5}$. Simulations show that the approximation errors are identical to those offered by the naive exhaustive method as given in Tables 6 and 7 and will not be repeated here.

5.5 Summary

In this chapter, we discussed various methods to improve the efficiency of KDE. Since KDE forms an integral part of our initialization algorithms and dominates the computational load, improving the efficiency of KDE allows for tremendous reduction in algorithm execution times. The first method we discussed focused on the particular choice of kernel function. Savings in computation were expected for two reasons: (i) choosing a kernel function that was simpler to evaluate, hence reducing the number of computations required during each call to KDE, and (ii) using finite extent kernels to prune data. It was shown that minimal savings in computation were offered by the particular choice of kernel function alone. The second method focused on restructuring the algorithm. By first resampling particles and then assigning the scaled weights, the order of computations required for KDE could be substantially reduced compared to the original implementation of the algorithm that assigned scaled weights first and then resampled particles. However, this saving in computation came at the cost of a loss in accuracy, which was expected since KDE performs better with larger data sets. The third, and most attractive method for reducing the computational load, was exploiting the geometry of the particle distribution using kd-trees. Sets of particles and importance weights were organized in multiple kd-trees and the dual-tree algorithm was executed on pairs of kd-trees taken in sequence. Kd-trees offered the greatest improvement in efficiency compared to the other methods under consideration, and the improved efficiency was achieved with no loss in accuracy.

The analysis on estimation accuracy given in this chapter must be taken with a grain of salt. With reference to the network, estimation accuracy is affected by various factors such as the accuracy of the individual sensors, sensor placement, number of sensors, types of sensors, state space being monitored and number of targets. With reference to the algorithm, estimation accuracy is dependent on factors such as the number of particles, individual nodes' organic state spaces, kernel function, kernel bandwidth and structure of the algorithm being used. With all algorithmic parameters kept constant, the particular network structure could completely change the level of accuracy achieved. Therefore, it is difficult, if not impossible, to select a set of parameters required to achieve a desired level of estimation accuracy in every scenario. Such a selection of parameters will vary across scenarios and choices are generally made subjectively. However, all other factors kept constant, the effect of varying one parameter on the estimation accuracy gives a good idea of the relative effect of that parameter. Therefore, the estimation errors presented herein should only be used for a comparative analysis and not considered in an absolute sense.

CHAPTER 6

CLOSING REMARKS

6.1 Contributions of the Thesis

In this dissertation, we have tackled the problem of initializing sensor networks in a fully distributed manner. The general formulation finds direct applicability to arbitrary sensor networks with multiple types of sensors. The network, as a whole, can be made capable of observing the state space of interest, even though individual sensors may not be capable of observing it locally. The initialization algorithm is built on the concepts of stochastic filtering, importance sampling, kernel density estimation, and a novel weighting and resampling strategy. We initially developed our algorithm for sensor networks using a simple chain communication topology. This simple communication topology allowed us to focus on the distributed data fusion aspects of the problem, separating it from the details of network connectivity. Several versions of the initialization algorithm were developed; some trade communication load with computation load, whereas others have the ability to compensate for time delays. For applicability to arbitrary sensor networks, we generalized our initialization algorithm to sensor networks with a tree communication topology. Comparisons with BP methods for sensor network initialization showed that our initialization algorithm provided greater robustness to missed detections, and changing communication topology and scheduling. Finally, we investigated various methods to reduce the computational load of the initialization algorithm, and found that kd-trees stood out as the most effective and promising solution.

6.2 Avenues for Future Work

Although we have addressed the theoretical formulation of the initialization algorithm in great detail, the problem is far from being solved in terms of a real world implementation. This opens up doors to new and exciting areas of research, some of which are discussed in

this section.

For the generalized implementation of the algorithm that is applicable to networks using a tree communication topology, a crucial question is how does one establish the communication structure? From a single node's perspective, all that needs to be defined is a parent node, if any, and a set of child nodes. However, from the network's perspective, this can become complicated. A simple technique would be to designate a particular node as the root, and establish a fixed spanning tree that is optimal in some sense. However, this is not a flexible implementation because the pre-selected root node might not necessarily be the first node to detect a target. If another node in the network is the first to detect a target, we would like that node to become the root of the tree. With a new root node, however, the optimal spanning tree might change. This motivates the need to develop methods to dynamically select a root node and establish the appropriate spanning tree for inter-node communication. This might be as simple as sending routing information with each message as it propagates through the network. Other ideas for such a flexible implementation might be borrowed from packet switched networks.

With the added flexibility in selecting the spanning tree and the root node as discussed above, one issue that needs to be addressed is what happens if multiple nodes launch the initialization algorithm simultaneously? In this case, multiple nodes designate themselves as the root nodes and generate their own spanning trees on the fly as messages propagate through the network. When these messages collide at possibly multiple sensors in the network, care must be taken when fusing data. There are two naive methods that come to mind. The first naive method would be to establish priority for incoming messages and only pass the messages with higher priority. Priority could be established based on various factors, such as the number of nodes represented in an incoming message, or the time when the message was first generated by a root node. The other messages with lower priority would be rejected. This is not ideal because useful knowledge contained within multiple received messages is discarded. Another naive method would be to simply allow each

node to run several independent instances of the initialization algorithm and then combine the final posterior distributions generated by each instance. Such an implementation is not ideal either because there is a communication and computation overhead of running multiple instances of the initialization algorithm. Processing time would also be a factor. In the generalized initialization algorithm for tree networks, parallel processing could occur at nodes that lie within the same level of the tree, but the fusion of data must occur sequentially as messages propagate between parent and child nodes. Therefore, the execution of a single instance of the initialization algorithm is not instantaneous. With multiple instances of the initialization algorithm running simultaneously at each node, the time required to generate the joint network distribution would only increase. A good method to eliminate the overhead and to handle such collisions would be a hybrid method that combines both naive methods described above. In such a method, after messages collide at a node, they would all be fused together and only one instance of the initialization algorithm would survive. This can become quite complicated depending on how large a tree has grown before it has been eliminated, and the number of collisions that occur. Work done in this area could lead to a dissertation by itself and it paves the road for exciting future work.

So far, we have only considered the initialization algorithm and focused on the distributed initialization block of our smart sensor nodes, the block diagram for which is once again shown in Figure 49. For a description of the functionality of each block, please refer to Section 1.1. The goal of the distributed initialization block was to acquire organic state

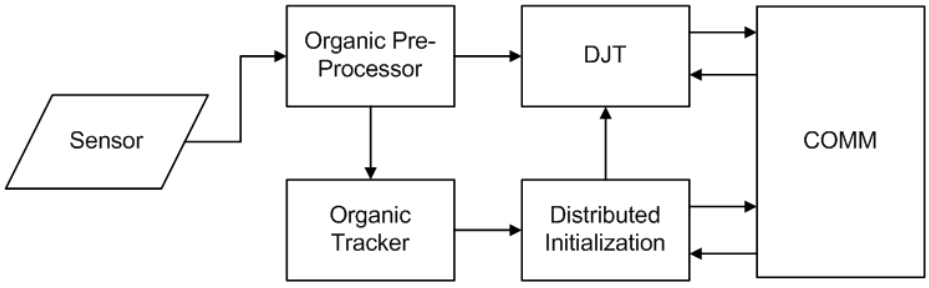


Figure 49. System block diagram of a smart sensor node.

estimates for new targets from the organic tracker, generate estimates for these new targets

in the target state space, and provide these estimates to the Distributed Joint Tracker (DJT). For seamless operation in a real world application, some handshaking between the organic tracker and the DJT is required to ensure that the initialization algorithm is launched if and only if it is required. The implementation is straightforward when no targets are being tracked by either tracker. In this case, any new target detected by the organic tracker is initialized in the target state space and fed into the DJT. However, if multiple targets are already being tracked by both the organic tracker and the DJT, coordinating the birth of new tracks and death of existing tracks is necessary. Consider a scenario in which multiple targets are being tracked by both the organic tracker and the DJT at the current node. Assume that the organic tracker now loses one target, either because it is occluded, or because it moves out of the detection range for that node. If all the other sensors in the network also lose the same target, then the DJT would no longer be capable of tracking it, and it would be responsible for killing the track. However, if some of the other sensors can still detect the target, then the DJT might be able to continue tracking it. Now assume that the target is once again detected by the organic tracker at the current node. How does the node know that this new target is already being tracked by the DJT? If the current node simply launches the initialization algorithm based on the new detection, we run the risk of having multiple tracks of the same target in the DJT. This would waste computation and communication, both of which are typically scarce resources in sensor networks.

The work presented in this dissertation has addressed the important issue of initialization for autonomous distributed target tracking systems in sensor networks. The relative infancy of this area of research provides countless avenues for improvement, some of which have been addressed in this chapter.

APPENDIX A

LOW COMPLEXITY INITIALIZATION ALGORITHM

In this appendix, we provide pseudocode for the low complexity initialization algorithm for sensor networks using the chain communication topology. The algorithm is developed in Section 2.2. Three communication passes are required for initializing the network: a forward pass to sequentially generate the particle support, a reverse pass to disseminate the final particles throughout the network and to sequentially generate importance weights, and a second forward pass to disseminate the final importance weights throughout the network. Data processing occurs only in the first two communication passes.

- **Variables:**

  $\mathbf{s}_t^{(i)}$ = particle i at time t

  $w_t^{(i)}$ = importance weight of particle $\mathbf{s}_t^{(i)}$

  $\mathbf{z}_{m,t}$ = the organic state estimate at node m at time t

  $\mathbf{z}_t = \{\mathbf{z}_{1,t}, \ldots, \mathbf{z}_{M,t}\}$

  D = number of particles used for initialization

  M = total number of nodes

  n_s = number of nodes that provided input to the algorithm

  $w_{\text{num}}^{(i)}$ = numerator of the importance weights

  $w_{\text{den}}^{(i)}$ = denominator of the importance weights

  $\check{w}_*$ = resampling weight

- **Forward Pass: Sequentially Generate Particles**

  $n_s = 0$

  Node 1:

  - If there is a detection
    - Generate D particles

· $\mathbf{s}_t^{(i)} \sim p\left(\mathbf{s}_t|\mathbf{z}_{1,t}\right),\ i = 1, \ldots, D$

* $n_s = 1$

– Else

* $\mathbf{s}_t^{(i)} = 0,\ i = 1, \ldots, D$

– Send $\{\mathbf{s}_t^{(i)}\}_{i=1}^{D}$ and n_s to Node 2

For *Node m*, $m = 2, \ldots M$:

– Receive $\{\mathbf{s}_t^{(i)}\}_{i=1}^{D}$ and n_s from Node $(m-1)$

– If there is a detection

* Label the received particles $\{\mathbf{s}_{r,t}^{(i)}\}_{i=1}^{D}$

* Assign resampling weights to received particles

· $\check{w}_{r,t} = n_s$

* Generate D new particles

· $\mathbf{s}_{n,t}^{(i)} \sim p\left(\mathbf{s}_t|\mathbf{z}_{m,t}\right),\ i = 1, \ldots, D$

* Assign resampling weights to new particles

· $\check{w}_{n,t} = 1$

* From the $2D$ particles, obtain D particles $\{\mathbf{s}_t^{(i)}\}_{i=1}^{D}$ by performing a weighted sampling with replacement according to the resampling weights

* $n_s = n_s + 1$

– If $m < M$

* Send $\{\mathbf{s}_t^{(i)}\}_{i=1}^{D}$ and n_s to Node $(m+1)$

- **Reverse Pass: Disseminate Particles and Sequentially Generate Importance Weights**

For *Node m*, $m = M, \ldots, 1$:

– If $m < M$

    - Accept $\{\mathbf{s}_t^{(i)},\ w_{\text{num}}^{(i)},\ w_{\text{den}}^{(i)}\}_{i=1}^{D}$ from Node $(m+1)$

  - Else

    - $w_{\text{num}}^{(i)} = 1,\ i = 1, \ldots, D$

    - $w_{\text{den}}^{(i)} = 0,\ i = 1, \ldots, D$

  - Update components of the importance weights

    - $w_{\text{num}}^{(i)} = w_{\text{num}}^{(i)} \cdot p(\mathbf{z}_{m,t}|\mathbf{s}_t^{(i)}),\ i = 1, \ldots, D$

    - $w_{\text{den}}^{(i)} = w_{\text{den}}^{(i)} + \frac{p(\mathbf{z}_{m,t}|\mathbf{s}_t^{(i)})}{p(\mathbf{z}_{m,t})},\ i = 1, \ldots, D$

  - If $m > 1$

    - Send $\{\mathbf{s}_t^{(i)},\ w_{\text{num}}^{(i)},\ w_{\text{den}}^{(i)}\}_{i=1}^{D}$ to Node $(m-1)$

- **Forward Pass: Disseminate Importance Weights**

  Node 1:

  - Determine the importance weights

    - $w_t^{(i)} = \frac{w_{\text{num}}^{(i)}}{w_{\text{den}}^{(i)}},\ i = 1, \ldots, D$

  - Normalize the importance weights

    - $w_t^{(i)} = \frac{w_t^{(i)}}{\sum_{i=1}^{D} w_t^{(i)}},\ i = 1, \ldots, D$

  - Send $\{w_t^{(i)}\}_{i=1}^{D}$ to Node 2

  For *Node m*, $m = 2, \ldots M$:

  - Accept $\{w_t^{(i)}\}_{i=1}^{D}$ from Node $(m-1)$

  - If $m < M$

    - Send $\{w_t^{(i)}\}_{i=1}^{D}$ to Node $(m+1)$

APPENDIX B

LOW LATENCY INITIALIZATION ALGORITHM

In this appendix, we provide pseudocode for the low latency initialization algorithm for sensor networks using the chain communication topology. The algorithm is developed in Section 2.3. Two communication passes are required for initializing the network: a forward pass to sequentially generate the particle support and importance weights, and a reverse pass to disseminate the final particles and importance weights throughout the network. Compared to the low complexity initialization algorithm, the low latency algorithm comes at a cost of increased computation at each node but is attractive if the latency incurred as a result of three communication passes is not acceptable.

- **Variables:**

  $\mathbf{s}_t^{(i)}$ = particle i at time t

  $w_t^{(i)}$ = importance weight of particle $\mathbf{s}_t^{(i)}$

  $\mathbf{z}_{m,t}$ = the organic state estimate at node m at time t

  $\mathbf{z}_t = \{\mathbf{z}_{1,t}, \ldots, \mathbf{z}_{M,t}\}$

  D = number of particles used for initialization

  M = total number of nodes

  n_s = number of nodes that provided input to the algorithm

  $\check{w}_*$ = resampling weight

  $W(\cdot)$ = kernel used for density estimation

- **Forward Pass: Sequentially Generate Particles and Weights**

  $n_s = 0$

  Node 1:

  - If there is a detection,
    - Generate D particles

· $\mathbf{s}_t^{(i)} \sim p\left(\mathbf{s}_t|\mathbf{z}_{1,t}\right), i = 1, \ldots, D$

* Assign importance weights

· $w_t^{(i)} = \frac{1}{D},\ i = 1, \ldots, D$

* $n_s = 1$

– Else,

* $\mathbf{s}_t^{(i)} = 0,\ i = 1, \ldots, D$

* $w_t^{(i)} = 0,\ i = 1, \ldots, D$

– Send $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ and n_s to Node 2

For *Node m*, $m = 2, \ldots M$

– Receive $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ and n_s from Node $(m-1)$

* These particles are distributed according to $\frac{1}{n_s}\sum_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_t|\mathbf{z}_{\tilde{m},t}\right)$

* Particles and weights together represent $p\left(\mathbf{s}_t|\mathbf{z}_{1,t}, \mathbf{z}_{2,t}, \ldots, \mathbf{z}_{m-1,t}\right) \propto \prod_{\tilde{m}=1}^{m-1} p\left(\mathbf{s}_t|\mathbf{z}_{\tilde{m},t}\right)$

– If there is a detection

* Label the received particles and weights $\{\mathbf{s}_{r,t}^{(i)},\ w_{r,t}^{(i)}\}_{i=1}^{D}$

* Generate D new particles

· $\mathbf{s}_{n,t}^{(i)} \sim p\left(\mathbf{s}_t|\mathbf{z}_{m,t}\right),\ i = 1, \ldots, D$

* Determine scaled weights

· $\tilde{w}_{n,t}^{(i)} = p\left(\mathbf{s}_{n,t}^{(i)}|\mathbf{z}_{m,t}\right) \cdot \sum_{j=1}^{D} w_{r,t}^{(j)} W\left(\mathbf{s}_{n,t}^{(i)} - \mathbf{s}_{r,t}^{(j)}\right),\ i = 1, \ldots, D$

· $\tilde{w}_{r,t}^{(i)} = p\left(\mathbf{s}_{r,t}^{(i)}|\mathbf{z}_{m,t}\right) \cdot \sum_{j=1}^{D} w_{r,t}^{(j)} W\left(\mathbf{s}_{r,t}^{(i)} - \mathbf{s}_{r,t}^{(j)}\right),\ i = 1, \ldots, D$

* From the $2D$ pairs of particles and scaled weights, sample D pairs as follows

· Assign resampling weight $\check{w}_{r,t} = n_s$ to $\{\mathbf{s}_{r,t}^{(i)},\ \tilde{w}_{r,t}^{(i)}\}_{i=1}^{D}$

· Assign resampling weight $\check{w}_{n,t} = 1$ to $\{\mathbf{s}_{n,t}^{(i)},\ \tilde{w}_{n,t}^{(i)}\}_{i=1}^{D}$

· Perform a weighted sampling with replacement according to the re-sampling weights $\check{w}_*$ to generate the set $\{\mathbf{s}_t^{(i)},\ \tilde{w}_t^{(i)}\}_{i=1}^{D}$

* Modify the scaled weights to account for the particle support

· $w_t^{(i)} = \frac{\tilde{w}_t^{(i)}}{\sum_{j=1}^{D} W\left(\mathbf{s}_t^{(i)} - \mathbf{s}_t^{(j)}\right)},\ i = 1, \ldots, D$

* Normalize the importance weights

· $w_t^{(i)} = \frac{w_t^{(i)}}{\sum_{j=1}^{D} w_t^{(j)}}, i = 1, \ldots, D$

* Final set of particles are distributed according to $\sum_{\tilde{m}=1}^{m} p\left(\mathbf{s}_t^{(i)} | \mathbf{z}_{\tilde{m},t}\right)$

* Final particles and importance weights together represent $p\left(\mathbf{s}_t^{(i)} | \mathbf{z}_{1,t}, \mathbf{z}_{2,t}, \ldots, \mathbf{z}_{m,t}\right) \propto \prod_{\tilde{m}=1}^{m} p\left(\mathbf{s}_t^{(i)} | \mathbf{z}_{\tilde{m},t}\right)$

* $n_s = n_s + 1$

– If $m < M$

* Send $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ and n_s to Node $(m + 1)$

- **Reverse Pass: Disseminate Particles and Importance Weights**

Node M:

– Send $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ to Node $(M - 1)$

For *Node m*, $m = M - 1, \ldots, 1$

– Receive $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ from Node $(m + 1)$

– if $m > 1$

* Send $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ to Node $(m - 1)$

APPENDIX C

INITIALIZATION ALGORITHM FOR TREE COMMUNICATION

In this appendix, we provide pseudocode for the generalized initialization algorithm for sensor networks using the tree communication topology. The algorithm is developed in Chapter 3. Three communication passes are required for initializing the network: a root-to-leaf pass, or downward pass, to sequentially generate the particle support and importance weights representing the joint distribution for individual branches, a leaf-to-root pass, or upward pass, to fuse distributions across branches, and a final downward pass to disseminate the final particles and importance weights throughout the network. Data processing occurs only in the first two communication passes. We assume that node 1 is the root of the tree.

- **Variables:**

  $\mathbf{s}_t^{(i)}$ = particle i at time t

  $w_t^{(i)}$ = importance weight of particle $\mathbf{s}_t^{(i)}$

  $\mathbf{z}_{m,t}$ = the organic state estimate at node m at time t

  $\mathbf{z}_t = \{\mathbf{z}_{1,t}, \ldots, \mathbf{z}_{M,t}\}$

  D = number of particles used for initialization

  M = total number of nodes

  $\dot{n}_s$ = number of nodes that provided input to the algorithm

  $\breve{w}_*$ = resampling weight

  $W(\cdot)$ = kernel used for density estimation

- **Downward Pass: Fuse Knowledge Over Branches of the Tree**

  $\dot{n}_s = 0$

  Node 1:

  - If there is a detection,

* Generate D particles
  - $\mathbf{s}_t^{(i)} \sim p\left(\mathbf{s}_t|\mathbf{z}_{1,t}\right), i = 1, \ldots, D$
* Assign importance weights
  - $w_t^{(i)} = \frac{1}{D},\ i = 1, \ldots, D$
* $\dot{n}_s = 1$

– Else,

* $\mathbf{s}_t^{(i)} = 0,\ i = 1, \ldots, D$
* $w_t^{(i)} = 0,\ i = 1, \ldots, D$

– Send $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ and $\dot{n}_s$ to all child nodes

For *Node m*, $m = 2, \ldots M$

– Receive $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ and $\dot{n}_s$ from parent node

– If there is a detection

* Label the received particles and weights $\{\mathbf{s}_{r,t}^{(i)},\ w_{r,t}^{(i)}\}_{i=1}^{D}$
* Generate D new particles
  - $\mathbf{s}_{n,t}^{(i)} \sim p\left(\mathbf{s}_t|\mathbf{z}_{m,t}\right),\ i = 1, \ldots, D$
* Determine scaled weights
  - $\tilde{w}_{n,t}^{(i)} = p\left(\mathbf{s}_{n,t}^{(i)}|\mathbf{z}_{m,t}\right) \cdot \sum_{j=1}^{D} w_{r,t}^{(j)} W\left(\mathbf{s}_{n,t}^{(i)} - \mathbf{s}_{r,t}^{(j)}\right),\ i = 1, \ldots, D$
  - $\tilde{w}_{r,t}^{(i)} = p\left(\mathbf{s}_{r,t}^{(i)}|\mathbf{z}_{m,t}\right) \cdot \sum_{j=1}^{D} w_{r,t}^{(j)} W\left(\mathbf{s}_{r,t}^{(i)} - \mathbf{s}_{r,t}^{(j)}\right),\ i = 1, \ldots, D$
* From the $2D$ pairs of particles and scaled weights, sample D pairs as follows
  - Assign resampling weight $\check{w}_{r,t} = \dot{n}_s$ to $\{\mathbf{s}_{r,t}^{(i)},\ \tilde{w}_{r,t}^{(i)}\}_{i=1}^{D}$
  - Assign resampling weight $\check{w}_{n,t} = 1$ to $\{\mathbf{s}_{n,t}^{(i)},\ \tilde{w}_{n,t}^{(i)}\}_{i=1}^{D}$
  - Perform a weighted sampling with replacement according to the resampling weights $\check{w}_*$ to generate the set $\{\mathbf{s}_t^{(i)},\ \tilde{w}_t^{(i)}\}_{i=1}^{D}$

* Modify the scaled weights to account for the particle support
  - $w_t^{(i)} = \frac{\tilde{w}_t^{(i)}}{\sum_{j=1}^{D} W\left(\mathbf{s}_t^{(i)} - \mathbf{s}_t^{(j)}\right)},\ i = 1, \ldots, D$
* Normalize the importance weights
  - $w_t^{(i)} = \frac{w_t^{(i)}}{\sum_{j=1}^{D} w_t^{(j)}}, i = 1, \ldots, D$
* $\dot{n}_s = \dot{n}_s + 1$

– If child nodes exist

* Send $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ and $\dot{n}_s$ to child nodes

- **Upward Pass: Fuse Knowledge Across Branches of the Tree**

For *Node m*, $m = M, \ldots, 1$

– If no child nodes exist

* Send $\{\mathbf{s}_t^{(i)},\ w_t^{(i)}\}_{i=1}^{D}$ and $\dot{n}_s$ to parent node

– Else

* Receive data from each child node
  - $\{\mathbf{s}_{\mathrm{C}_n,t}^{(i)},\ w_{\mathrm{C}_n,t}^{(i)}\}_{i=1}^{D}$ and $\dot{n}_{\mathrm{C}_n}$ are received from child $\mathrm{C}_n,\ n = 1, \ldots, N$
* Fused knowledge from the downward pass is still stored at node *m*
  - $\{\hat{\mathbf{s}}_t^{(i)},\ \hat{w}_t^{(i)}\}_{i=1}^{D}$ and $\hat{n}_s$ represents prior knowledge from downward pass
* Determine scaled weights
  - $$\tilde{w}_{\mathrm{C}_n,t}^{(i)} = \frac{\prod_{\bar{n}=1}^{N}\left(\sum_{j=1}^{D} w_{\mathrm{C}_{\bar{n}},t}^{(j)} W\left(\mathbf{s}_{\mathrm{C}_n,t}^{(i)} - \mathbf{s}_{\mathrm{C}_{\bar{n}},t}^{(j)}\right)\right)}{\left(\sum_{j=1}^{D} \hat{w}_t^{(j)} W\left(\mathbf{s}_{\mathrm{C}_n,t}^{(i)} - \hat{\mathbf{s}}_t^{(j)}\right)\right)^{N-1}},\ i = 1, \ldots, D,\ n = 1, \ldots, N$$
* From the ND pairs of particles and scaled weights, sample D pairs as follows
  - Particles representing successor posteriors only are given by $\{\mathbf{s}_{\mathrm{C}_n,\mathrm{C},t}^{(j)}\} = \{\mathbf{s}_{\mathrm{C}_n,t}^{(i)}\}_{i=1}^{D} \bigcap \overline{\{\hat{\mathbf{s}}_t^{(i)}\}_{i=1}^{D}},\ n = 1, \ldots, N$

and resampling weights are assigned as

$\{\check{w}^{(j)}_{\mathrm{C}_n,\mathrm{C},t}\} = \dot{n}_{\mathrm{C}_n},\ n = 1,\ldots,N$

· Particles representing ancestor posteriors only are given by

$\{\mathbf{s}^{(j)}_{\mathrm{C}_n,\mathrm{P},t}\} = \{\mathbf{s}^{(i)}_{\mathrm{C}_n,t}\}_{i=1}^{D} \bigcap \{\hat{\mathbf{s}}^{(i)}_{t}\}_{i=1}^{D},\ n = 1,\ldots,N$

and resampling weights are assigned as

$\{\check{w}^{(j)}_{\mathrm{C}_n,\mathrm{P},t}\} = \frac{\dot{n}_{\mathrm{C}_n}}{N},\ n = 1,\ldots,N$

· Perform a weighted sampling with replacement according to the resampling weights $\check{w}_*$ to generate the set $\{\mathbf{s}^{(i)}_t,\ \tilde{w}^{(i)}_t\}_{i=1}^{D}$

∗ Modify the scaled weights to account for the particle support

· $w^{(i)}_t = \frac{\tilde{w}^{(i)}_t}{\sum_{j=1}^{D} W\left(\mathbf{s}^{(i)}_t - \mathbf{s}^{(j)}_t\right)},\ i = 1,\ldots,D$

∗ Normalize the importance weights

· $w^{(i)}_t = \frac{w^{(i)}_t}{\sum_{j=1}^{D} w^{(j)}_t}, i = 1,\ldots,D$

∗ $\dot{n}_s = \left(\sum_{n=1}^{N} \dot{n}_{\mathrm{C}_n}\right) - (N-1)\hat{n}_s$

– If node m is not the root node

∗ Send $\{\mathbf{s}^{(i)}_t,\ w^{(i)}_t\}_{i=1}^{D}$ and $\dot{n}_s$ to the parent node

- **Reverse Pass: Disseminate Particles and Importance Weights**

Node 1:

– Send $\{\mathbf{s}^{(i)}_t,\ w^{(i)}_t\}_{i=1}^{D}$ to child nodes

For *Node m*, $m = 2,\ldots,M$

– Receive $\{\mathbf{s}^{(i)}_t,\ w^{(i)}_t\}_{i=1}^{D}$ from parent node

– if child nodes exist

∗ Send $\{\mathbf{s}^{(i)}_t,\ w^{(i)}_t\}_{i=1}^{D}$ to child nodes

REFERENCES

[1] H. Gharavi and S. Kumar, eds., *Special issue on sensor networks and applications*. Proceedings of the IEEE, 2003.

[2] D. Goldberg, A. Andreou, P. Julián, P. Pouliquen, L. Riddle, and R. Rosasco, "A wake-up detector for an acoustic surveillance sensor network: algorithm and VLSI implementation," in *Proceedings of the International Conference on Information Processing in Sensor Networks*, pp. 134–141, 2004.

[3] J. Chamberland and V. Veeravalli, "Decentralized detection in sensor networks," *IEEE Transactions on Signal Processing*, vol. 51, no. 2, pp. 407–416, 2003.

[4] J. Li and R. Chellappa, "A factorization method for structure from planar motion," *Proceedings of the IEEE Workshop on Motion and Video Computing*, pp. 154–159, 2005.

[5] V. Cevher, A. Sankaranarayanan, J. McClellan, and R. Chellappa, "Target tracking using a joint acoustic video system," *IEEE Transactions on Multimedia*, vol. 9, no. 4, pp. 715 – 727, 2007.

[6] Y. Bar-Shalom and T. Fortmann, *Tracking and Data Association*. Academic-Press, 1988.

[7] A. Doucet, "On sequential simulation-based methods for Bayesian filtering," Tech. Rep. CUED/F-INFENG/TR.310, Department of Engineering, University of Cambridge, 2001.

[8] B. Rao, H. Durrant-Whyte, and J. Sheen, "A fully decentralized multi-sensor system for tracking and surveillance," *The International Journal of Robotics Research*, vol. 12, no. 1, p. 20, 1993.

[9] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, 2002.

[10] J. Hammersley and D. Handscomb, *Monte Carlo Methods*. Methuen, 1964.

[11] R. Snelick, U. Uludag, A. Mink, M. Indovina, and A. Jain, "Large scale evaluation of multimodal biometric authentication using state-of-the-art systems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 450–455, 2005.

[12] G. Pottie and W. Kaiser, "Wireless integrated network sensors," *Communications of the ACM*, vol. 43, pp. 51–58, May 2000.

[13] M. Coates, "Distributed particle filtering for sensor networks," *International Symposium on Information Processing in Sensor Networks*, 2004.

[14] R. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[15] H. Sorenson, *Kalman Filtering: Theory and Application*. IEEE Press, 1985.

[16] H. Tanizaki and R. Mariano, "Prediction, filtering and smoothing in non-linear and non-normal cases using Monte Carlo integration," *Journal of Applied Econometrics*, vol. 9, pp. 163–179, 1994.

[17] W. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, p. 97, 1970.

[18] M. Johns, "Importance sampling for bootstrap confidence intervals," *Journal of the American Statistical Association*, vol. 83, no. 403, pp. 709–714, 1988.

[19] B. Efron, "Bootstrap methods: Another look at the jackknife," *The Annals of Statistics*, vol. 7, no. 1, pp. 1–26, 1979.

[20] B. Ripley, *Stochastic Simulation*. John Wiley & Sons Inc., 1987.

[21] H. Trotter and J. Tukey, "Conditional Monte Carlo for normal samples," in *Symposium on Monte Carlo Methods*, pp. 64–79, 1956.

[22] A. Doucet, N. Freitas, and N. Gordon, eds., *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

[23] E. Fix and J. Hodges, "Discriminatory analysis, nonparametric estimation: consistency properties," Tech. Rep. 4, Project 21-49-004, USAF School of Aviation Medicine, Randolph Field, TX, 1951.

[24] B. Silverman, *Density estimation for statistics and data analysis*. Chapman and Hall, 191986.

[25] J. Munkres, *Analysis on Manifolds*. Perseus Books, 1990.

[26] M. Isard and A. Blake, "Condensation – conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, pp. 5–28, 1998.

[27] M. Isard, "PAMPAS: real-valued graphical models for computer vision," in *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2003.

[28] V. Cevher, R. Chellappa, F. Shah, R. Velmurugan, and J. H. McClellan, "An acoustic multi-target tracking system using random sampling consensus," in *2007 IEEE Aerospace Conference*, (Big Sky, Montana), 3–10 March 2007.

[29] V. Cevher, R. Velmurugan, and J. H. McClellan, "A range-only multiple target particle filter tracker," in *2006 IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Toulouse, France), May 2006.

[30] V. Cevher and J. H. McClellan, "General direction-of-arrival tracking with acoustic nodes," *IEEE Transactions on Signal Processing*, vol. 53, pp. 1–12, Jan. 2005.

[31] R. Allen and S. Blackman, "Implementation of an angle-only tracking filter," in *Proceedings of the SPIE*, vol. 1481, pp. 292–303, 1991.

[32] A. Farina, "Target tracking with bearings-only measurements," *Elsevier Signal Processing*, vol. 78, pp. 61–78, 1999.

[33] Y. Zhou, P. Yip, and H. Leung, "Tracking the direction-of-arrival of multiple moving targets by passive arrays: Algorithm," *IEEE Transactions on Signal Processing*, vol. 47, no. 10, pp. 2655–2666, October 1999.

[34] J. Sanchez-Araujo and S. Marcos, "An efficient PASTd-algorithm implementation for multiple direction of arrival tracking," *IEEE Transactions on Signal Processing*, vol. 47, pp. 2321–2324, August 1999.

[35] V. Aidala, "Kalman filter behavior in bearings-only tracking applications," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-15, pp. 29–39, January 1979.

[36] E. Brookner, *Tracking and Kalman Filtering Made Easy*. John Wiley and Sons, 1988.

[37] P. R. Kalata and K. M. Murphy, "$\alpha - \beta$ target tracking with track rate variations," in *IEEE Proceedings of the Twenty-Ninth Southeastern Symposium on Systems Theory*, 9-11 March 1997.

[38] Y. Weiss and W. Freeman, "Correctness of belief propagation in Gaussian graphical models of arbitrary topology," *Neural Computation*, vol. 13, no. 10, pp. 2173–2200, October 2001.

[39] J. Johnson and A. Willsky, "Walk-sum interpretation and analysis of Gaussian belief propagation," in *Advances in Neural Information Processing Systems*, (Vancouver, B.C., Canada), December 2006.

[40] M. Cetin, L. Chen, J. F. III, A. Ihler, R. Moses, M. Wainwright, and A. Willsky, "Distributed data fusion in sensor networks," *IEEE Signal Processing Magazine*, vol. 23, no. 4, pp. 42–55, July 2006.

[41] J. Yedidia, W. Freeman, and Y. Weiss, "Generalized belief propagation," *Advances in Neural Information Processing Systems*, vol. 13, pp. 689–695, 2001.

[42] R. Chellappa and A. Jain, eds., *Markov Random Fields. Theory and application*. Boston: Academic Press, 1993, 1993.

[43] T. Cover and J. Thomas, *Elements of Information Theory*. Wiley New York, 1991.

[44] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky, "Nonparametric belief propagation," in *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2003.

[45] A. Ihler, E. Sudderth, W. Freeman, and A. Willsky, "Efficient multiscale sampling from products of Gaussian mixtures," in *Neural Information Processing Systems 17*, (Vancouver, British Columbia, Canada), 9-11 December 2003.

[46] A. Ihler, J. Fisher, R. Moses, and A. Willsky, "Nonparametric belief propagation for self-localization of sensor networks," *IEEE Journal on Selected Areas in Communication*, vol. 23, no. 4, pp. 809–819, April 2005.

[47] A. Gray and A. Moore, "Rapid evaluation of multiple density models," in *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, (Key West, FL), January 2003.

[48] V. Epanechnikov, "Nonparametric estimation of of a multidimensional probability density," *Theory of Probability and its Applications*, vol. 14, no. 1, pp. 153–158, 1969.

[49] J. Friedman, J. Bentley, and R. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions Mathematical Software*, vol. 3, no. 3, pp. 209–226, September 1977.

[50] K. Deng and A. Moore, "Multi-resolution instance based learning," in *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, (San Francisco, CA), pp. 1233–1239, 1995.

[51] A. Gray and A. Moore, "Nonparametric density estimation: toward computational tractability," in *Proceedings of the Third SIAM International Conference on Data Mining*, (San Francisco, CA), 2003.

[52] A. Ihler, "Kernel density estimation toolbox for Matlab." 2003. Available http://www.ics.uci.edu/ ihler/code/kde.php.

VITA

Milind Borkar was born in Mumbai, India, in 1980. He received his B.S. and M.S. degrees in Electrical and Computer Engineering at the Georgia Institute of Technology in 2002 and 2005 respectively. He is currently a Ph.D. student at the Center for Signal and Image Processing, Georgia Institute of Technology, under the supervision of Dr. James H. McClellan. He has been a researcher for the Advanced Sensors Collaborative Technology Alliance (Microsensor Technical Area) sponsored by the Army Research Laboratory. He is expected to receive his Ph.D. degree in 2007. He has extensive experience working in industry through multiple internships. During the course of his bachelor's degree program, he was employed by P.B.J. Industries in Mumbai, India, in 2000, and by Intel Corporation, Beaverton, Oregon, in 2001. During his graduate program, he was employed by Texas Instruments, Dallas, Texas, in 2006 and 2007 respectively. His research interests include Markov chain Monte Carlo methods, smart sensor networks, heterogeneous sensor fusion, personnel detection and tracking, digital predistortion for linearizing nonlinear systems, and the development of multimedia tools for education.